


```

LL                      IIIIII                      SSSSSSSS
LL                      IIIIII                      SSSSSSSS
LL                      II                      SS
LL                      II                      SS
LL                      II                      SS
LL                      II                      SS
LL                      II                      SSSSSS
LL                      II                      SSSSSS
LL                      II                      SS
LL                      II                      SS
LL                      II                      SS
LL                      II                      SS
LL                      II                      SS
LLLLLLLLLLLLLL          IIIIII                      SSSSSSSS
LLLLLLLLLLLLLL          IIIIII                      SSSSSSSS

```

MONMAIN: Procedure Returns(Fixed Binary(31))
 Options(Ident('V04-000'), Main);

```
1  MONMAIN: Procedure      Returns(Fixed Binary(31))
2
3
4  /*
5  /******
6  /*
7  /* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
8  /* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
9  /* ALL RIGHTS RESERVED.
10 /*
11 /* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
12 /* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
13 /* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
14 /* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
15 /* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
16 /* TRANSFERRED.
17 /*
18 /* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
19 /* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
20 /* CORPORATION.
21 /*
22 /* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
23 /* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
24 /*
25 /*
26 /******
27 /*/
28
29 /*
30 /*++
31 /* FACILITY: VAX/VMS MONITOR Utility
32 /*
33 /* ABSTRACT: MAIN Routine, including command interface.
34 /*
35 /*
36 /* ENVIRONMENT:
37 /*
38 /*            Unprivileged user mode,
39 /*            except for certain collection routines which
40 /*            run in EXEC or KERNEL mode to access system
41 /*            data bases.
42 /*
43 /* AUTHOR: Thomas L. Cafarella, April, 1981
44 /*
45
```

```
46 1 /*
47 1 /* MODIFIED BY:
48 1 /*
49 1 /* V03-018 TLC1089 Thomas L. Cafarella 26-Jul-1984 11:00
50 1 /* Accept a space character in time specification.
51 1 /*
52 1 /* V03-017 TLC1087 Thomas L. Cafarella 25-Jul-1984 15:00
53 1 /* Default to /ALL when summarizing.
54 1 /*
55 1 /* V03-016 TLC1075 Thomas L. Cafarella 27-Jun-1984 15:00
56 1 /* Add stickiness to /INPUT qualifier.
57 1 /*
58 1 /* V03-015 TLC1073 Thomas L. Cafarella 02-May-1984 13:00
59 1 /* Make MAX_INP_FILES limit bigger.
60 1 /*
61 1 /* V03-014 PRS1012 Paul R. Senn 23-Mar-1984 14:00
62 1 /* Add wildcard capability for MF summary.
63 1 /*
64 1 /* V03-013 TLC1056 Thomas L. Cafarella 23-Mar-1984 13:00
65 1 /* Exclude class which is disabled.
66 1 /*
67 1 /* V03-012 PRS1011 Paul R. Senn 29-Feb-1984 14:00
68 1 /* add /FLUSH_INTERVAL qualifier
69 1 /*
70 1 /* V03-011 TLC1052 Thomas L. Cafarella 17-Feb-1984 11:00
71 1 /* Add multi-file summary capability.
72 1 /*
73 1 /* V03-010 PRS1002 Paul R. Senn 29-Dec-1983 16:00
74 1 /* GLOBALDEF VALUE symbols must now be longwords;
75 1 /* Use %REPLACE rather than GLOBALDEF VALUE for any equated
76 1 /* symbols which are not 4 bytes in length;
77 1 /*
78 1 /* V03-010 PRS1001 Paul R. Senn 27-Dec-1983 16:00
79 1 /* Add ALL CLASSES Pseudo-class
80 1 /*
81 1 /* V03-009 TLC1044 Thomas L. Cafarella 24-Aug-1983 13:00
82 1 /* Eliminate CLI 'NOCOMD' error for comment lines.
83 1 /*
84 1 /* V03-008 SPC0007 Stephen P. Carney 24-Jun-1983 16:00
85 1 /* Add EXECUTE subcommand.
86 1 /*
87 1 /* V03-007 TLC1042 Thomas L. Cafarella 19-Jun-1983 15:00
88 1 /* Add /ITEM qualifier for homogeneous classes.
89 1 /*
90 1 /* V03-007 TLC1041 Thomas L. Cafarella 16-Jun-1983 15:00
91 1 /* Ignore CLI error message when no command on line.
92 1 /*
93 1 /* V03-007 TLC1038 Thomas L. Cafarella 14-Jun-1983 18:00
94 1 /* Make default list of classes replace previous list.
95 1 /*
96 1 /* V03-006 TLC1028 Thomas L. Cafarella 14-Apr-1983 16:00
97 1 /* Add interactive user interface.
98 1 /*
99 1 /* V03-005 TLC1019 Thomas L. Cafarella 18-Jun-1982 16:00
100 1 /* Change CLI$_NEGATED symbol to CLI$_LOCNEG.
101 1 /*
```

MONMAIN
V04-000

B 11
16-SEP-1984 02:10:49
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273 Page 3
DISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (2)

102	:	1	/*	V03-004	TLC1012	Thomas L. Cafarella	30-Mar-1982	13:00
103	:	1	/*		Display user's comment string on screen line 5.			
104	:	1	/*					
105	:	1	/*	V03-004	TLC1011	Thomas L. Cafarella	29-Mar-1982	20:00
106	:	1	/*		Move system service names for SSERROR msg to static storage.			
107	:	1	/*					
108	:	1	/*	V03-003	TLC1009	Thomas L. Cafarella	29-Mar-1982	01:00
109	:	1	/*		Get current time when other times are converted.			
110	:	1	/*					
111	:	1	/*	V03-003	TLC1007	Thomas L. Cafarella	28-Mar-1982	19:00
112	:	1	/*		Add checks for maximum sizes of qualifier values.			
113	:	1	/*					
114	:	1	/*	V03-002	TLC1003	Thomas L. Cafarella	23-Mar-1982	13:00
115	:	1	/*		Fix up module headers.			
116	:	1	/*					
117	:	1	/*	V03-001	TLC1001	Thomas L. Cafarella	16-Mar-1982	13:00
118	:	1	/*		Add CTRL-W screen refresh support.			
119	:	1	/*					
120	:	1	/*--					
121	:	1	/*/					
122	:	1						

```
123 | 1 | /*
124 | 1 | /*
125 | 1 | /*
126 | 1 | /*
127 | 1 | /*
128 | 1 | /*
129 | 1 | /*/
130 | 1 |
131 | 1 | %INCLUDE      MONDEF;      /* Monitor utility structure definitions */
899 | 1 | %INCLUDE      $CHFDEF;     /* Condition handler facility definitions */
919 | 1 | %INCLUDE      $STSDEF;     /* Status code definitions */
936 | 1 |
937 | 1 | /*
938 | 1 | /*
939 | 1 | /*
940 | 1 | /*
941 | 1 | /*
942 | 1 | /*
943 | 1 | /*/
944 | 1 |
945 | 1 |
```

INCLUDE FILES

SYSTEM SERVICE MACRO DEFINITIONS

```
946 | 1 | /*
947 | 1 | /*
948 | 1 | /*
949 | 1 | /*
950 | 1 | /*
951 | 1 | /*
952 | 1 | /*
953 | 1 | /*
954 | 1 | %REPLACE NOT_SUCCESSFUL BY '0'B; /* Failing status bit */
955 | 1 | %REPLACE YES BY '1'B; /* For general use */
956 | 1 | %REPLACE NO BY '0'B; /* For general use */
957 | 1 |
```

COMPILE-TIME CONSTANTS

```

958 | 1 /*
959 | 1 /*
960 | 1 /*
961 | 1 /*
962 | 1 /*
963 | 1 /*
964 | 1 /*/
965 | 1
966 | 1 Declare
967 | 1 MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
968 | 1
969 | 1 SIGNAL MON_ERR ENTRY, /* Routine to log synchronous errors */
970 | 1 DISPLAY_CLEANUP ENTRY RETURNS(FIXED BINARY(31)), /* Routine to signal MONITOR errors */
971 | 1 $$$_NORMAL FIXED BINARY(31) GLOBALREF VALUE, /* Procedure clean up DISPLAY processing */
972 | 1 RMSS_EOF FIXED BINARY(31) GLOBALREF VALUE, /* System normal return status */
973 | 1 RMSS_NMF FIXED BINARY(31) GLOBALREF VALUE, /* RMS end-of-file return status */
974 | 1 RMSS_FNF FIXED BINARY(31) GLOBALREF VALUE, /* RMS no-more-files message for wildcard parsing */
975 | 1 NORMAL FIXED BINARY(31) GLOBALDEF; /* RMS file-not-found message */
976 | 1 /* MONITOR normal return status */

```

977	1	Declare				
978	1	MNR\$_ERREXEREA	FIXED BINARY(31)	GLOBALREF	VALUE,	/* Error message code */
979	1	MNR\$_ERRREC FIL	FIXED BINARY(31)	GLOBALREF	VALUE,	/* Error message code */
980	1	MNR\$_UNEXPERR	FIXED BINARY(31)	GLOBALREF	VALUE,	/* Error message code */
981	1	MNR\$_ERRPARSE	FIXED BINARY(31)	GLOBALREF	VALUE,	/* Error message code */
982	1	CLIS_NOCOMD	FIXED BINARY(31)	GLOBALREF	VALUE;	/* CLI error message code */
983	1					

```
984 1 /*
985 1 /*
986 1 /*
987 1 /*
988 1 /*
989 1 /*
990 1 /*
991 1
992 1 Declare
993 1 COMMAND_FILE FILE GLOBALREF; /* Execute Command File */
994 1
995 1 Declare
996 1 CURR_ERRCODE FIXED BINARY(31) GLOBALDEF INIT(0), /* MONITOR error status code currently expected */
997 1 FIRST_MON_CMD BIT(1) ALIGNED GLOBALDEF INIT('0'B), /* YES => first MONITOR (DCL-level) cmd executing */
998 1 PROMPT BIT(1) ALIGNED GLOBALDEF INIT('0'B), /* YES => prompt user for another subcommand */
999 1 EXECUTE BIT(1) ALIGNED GLOBALDEF INIT('0'B), /* YES => read another command from the execute file
1000 1 DISPLAYING BIT(1) ALIGNED GLOBALDEF INIT('0'B); /* YES => terminal display output is active */
1001 1
1002 1 Declare
1003 1 CDBPTR POINTER GLOBALDEF, /* Pointer to CDB (Class Descriptor Block) */
1004 1 MRBPTR POINTER GLOBALDEF, /* Pointer to MRB (Monitor Request Block) */
1005 1 DEF_MRPTR POINTER GLOBALDEF, /* Pointer to "default" MRB (Monitor Request Block) */
1006 1 TEMP_MRPTR POINTER GLOBALDEF, /* Pointer to "temporary" MRB (Monitor Request Block) */
1007 1 CURR_MRPTR POINTER GLOBALDEF, /* Pointer to "current" MRB (Monitor Request Block) */
1008 1 ACT_MRPTR POINTER GLOBALDEF, /* Pointer to "active" MRB (Monitor Request Block) */
1009 1 MCAPTR POINTER GLOBALDEF, /* Pointer to MCA (Monitor Communication Area) */
1010 1 SPTR POINTER GLOBALDEF; /* Pointer to SYI (System Information Area) */
1011 1
1012 1 Declare
1013 1 QUALPTR POINTER GLOBALDEF, /* Pointer to Qualifier Descriptors Block */
1014 1 DEFPTR POINTER GLOBALDEF; /* Pointer to Qualifier Default Value Descriptors Block */
1015 1
1016 1 Declare
1017 1 STAT_TABLE GLOBALDEF, /* Table of pointers to str descs for statistic qualifiers
1018 1 2 STAT_DESC (0:STATS-1) POINTER;
1019 1
1020 1 Declare
1021 1 PROCD_TABLE GLOBALDEF, /* Table of pointers to str descs for PROCESSES display qua
1022 1 2 PROCD_DESC (0:PROCDISPS-1) POINTER;
1023 1
```

```

1024 1
1025 1
1026 1
1027 1
1028 1
1029 1
1030 1
1031 1
1032 1
1033 1
1034 1
1035 1
1036 1
1037 1
1038 1
1039 1
1040 1
1041 1
1042 1
1043 1
1044 1
1045 1
1046 1
1047 1
1048 1
1049 1

/*
/*      Counted strings for system service names used in the MNR$_SSERROR error message
/*
/*
Declare
1 READDEF_STR GLOBALDEF,
2 L      FIXED BINARY(7) INIT(7),
2 S      CHAR(7) INIT('$READDEF'),
/* Counted string for $READDEF */
/* Length */
/* String */

1 CLREF_STR GLOBALDEF,
2 L      FIXED BINARY(7) INIT(6),
2 S      CHAR(7) INIT('$CLREF'),
/* Counted string for $CLREF */
/* Length */
/* String */

1 SCHDWK_STR GLOBALDEF,
2 L      FIXED BINARY(7) INIT(7),
2 S      CHAR(7) INIT('$SCHDWK'),
/* Counted string for $SCHDWK */
/* Length */
/* String */

1 SETIMR_STR GLOBALDEF,
2 L      FIXED BINARY(7) INIT(7),
2 S      CHAR(7) INIT('$SETIMR'),
/* Counted string for $SETIMR */
/* Length */
/* String */

1 DCLAST_STR GLOBALDEF,
2 L      FIXED BINARY(7) INIT(7),
2 S      CHAR(7) INIT('$DCLAST'),
/* Counted string for $DCLAST */
/* Length */
/* String */

```

```
1050 : 1 /*
1051 : 1 /*
1052 : 1 /*
1053 : 1 /*
1054 : 1 /*
1055 : 1 /*
1056 : 1 /*
1057 : 1
1058 : 1 Declare
1059 : 1 CALL          FIXED BINARY(31),      /* Holds function value (return status) of called routines */
1060 : 1 STATUS      BIT(1)  BASED(ADDR(CALL)); /* Low-order status bit for called routines */
1061 : 1
1062 : 1 /*
1063 : 1 /*      Strings for command qualifiers and parameter. Descriptors
1064 : 1 /*      for these strings are defined in the QUAL structure.
1065 : 1 /*
1066 : 1
1067 : 1 DECLARE
1068 : 1
1069 : 1 BEG_QUAL_S      CHAR(9) INIT('BEGINNING'),      /* BEGINNING qualifier string */
1070 : 1 END_QUAL_S      CHAR(6) INIT('ENDING'),          /* ENDING qualifier string */
1071 : 1 INT_QUAL_S      CHAR(8) INIT('INTERVAL'),        /* INTERVAL qualifier string */
1072 : 1 FLUSH_QUAL_S    CHAR(14) INIT('FLUSH INTERVAL'), /* FLUSH INTERVAL qualifier string */
1073 : 1 VIEW_QUAL_S     CHAR(12) INIT('VIEWING_TIME'),   /* VIEWING_TIME qualifier string */
1074 : 1 INP_QUAL_S      CHAR(5) INIT('INPUT'),            /* INPUT qualifier string */
1075 : 1 DISP_QUAL_S     CHAR(7) INIT('DISPLAY'),         /* DISPLAY qualifier string */
1076 : 1 REC_QUAL_S      CHAR(6) INIT('RECORD'),          /* RECORD qualifier string */
1077 : 1 SUMM_QUAL_S     CHAR(7) INIT('SUMMARY'),          /* SUMMARY qualifier string */
1078 : 1 COMM_QUAL_S     CHAR(7) INIT('COMMENT'),         /* COMMENT qualifier string */
1079 : 1 BY_NODE_QUAL_S  CHAR(7) INIT('BY_NODE'),         /* BY_NODE qualifier string */
1080 : 1 CLASS_PARM_S    CHAR(10) INIT('CLASS_NAME');     /* CLASS_NAME parameter string */
1081 : 1
1082 : 1 /*
1083 : 1 /*      Strings for class-name qualifiers. Descriptors for these strings are defined in
1084 : 1 /*      the QUAL structure.
1085 : 1 /*
1086 : 1
1087 : 1 DECLARE
1088 : 1
1089 : 1 ALL_QUAL_S      CHAR(3) INIT('ALL'),              /* ALL qualifier string */
1090 : 1 CUR_QUAL_S      CHAR(7) INIT('CURRENT'),          /* CURRENT qualifier string */
1091 : 1 AVE_QUAL_S      CHAR(7) INIT('AVERAGE'),         /* AVERAGE qualifier string */
1092 : 1 MIN_QUAL_S      CHAR(7) INIT('MINIMUM'),          /* MINIMUM qualifier string */
1093 : 1 MAX_QUAL_S      CHAR(7) INIT('MAXIMUM'),          /* MAXIMUM qualifier string */
1094 : 1 TOPC_QUAL_S     CHAR(6) INIT('TOPCPU'),           /* TOPCPU qualifier string */
1095 : 1 TOPD_QUAL_S     CHAR(6) INIT('TOPDIO'),           /* TOPDIO qualifier string */
1096 : 1 TOPB_QUAL_S     CHAR(6) INIT('TOPBIO'),           /* TOPBIO qualifier string */
1097 : 1 TOPF_QUAL_S     CHAR(8) INIT('TOPFAULT'),         /* TOPFAULT qualifier string */
1098 : 1 CPU_QUAL_S      CHAR(3) INIT('CPU'),              /* CPU qualifier string */
1099 : 1 PCENT_QUAL_S    CHAR(7) INIT('PERCENT'),          /* PERCENT qualifier string */
1100 : 1 ITEM_QUAL_S     CHAR(4) INIT('ITEM');             /* ITEM qualifier string */
1101 : 1
1102 : 1
1103 : 1 /*
1104 : 1 /*      Default file-spec values for qualifiers. Descriptors for these strings are
1105 : 1 /*      defined in the DEF structure.
```

MONMAIN
V04-000

J 11
16-SEP-1984 02:10:52
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
DISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (9)
Page 11

1106 : 1
1107 1
1108 1
1109 1
1110 1
1111 1
1112 1
1113 1
1114 1

/*

DECLARE

REC_DEF_S
DISP_DEF_S
SUMM_DEF_S

CHAR(11) INIT('MONITOR.DAT'),
CHAR(11) INIT('SYSS\$OUTPUT:'),
CHAR(11) INIT('MONITOR.SUM');

/* RECORD qualifier default value */
/* DISPLAY qualifier default value */
/* SUMMARY qualifier default value */

```

1115 1
1116 1 DECLARE
1117 1
1118 1 01 MRB_DEF,
1119 1
1120 1      02 beg      bit(64) aligned      init('0'B), /* Beginning time of request in system time units */
1121 1      02 end      bit(64) aligned      init('0'B), /* Ending time of request in system time units */
1122 1      02 int      fixed binary(31)      init(0), /* Interval value in seconds */
1123 1      02 flush     fixed binary(31)      init(0), /* Flush interval in seconds */
1124 1      02 vie      fixed binary(31)      init(0), /* Viewing time for a screen in seconds */
1125 1      02 inp      pointer                init(null()), /* Address of input file descr (0 if input not requested) */
1126 1      02 dis      pointer                init(null()), /* Address of display file descr (0 if display not requested)
1127 1      ;                                     /* (Initialized in MONITOR_INIT routine) */
1128 1      02 rec      pointer                init(null()), /* Address of record file descr (0 if record not requested)
1129 1      02 sum      pointer                init(null()), /* Address of summary file descr (0 if summary not requested)
1130 1      02 com      pointer                init(null()), /* Address of comment string descriptor */
1131 1      02 clc      fixed binary(15),      /* Count of classes requested (needs no init) */
1132 1      02 clb      bit(128) aligned      init('0'B), /* Bit string of requested classes */
1133 1      02 inf      fixed binary(7),      /* Count of input files specified (needs no init) */
1134 1      02 fla,
1135 1          03 dis     bit(1)                init(NO), /* Flags for MRB */
1136 1          03 rec     bit(1)                init(NO),
1137 1          03 sum     bit(1)                init(NO),
1138 1          03 pbk     bit(1)                init(NO),
1139 1          03 ide     bit(1)                init(NO),
1140 1          03 dtf     bit(1)                init(NO),
1141 1          03 icr     bit(1)                init(NO),
1142 1          03 rcr     bit(1)                init(NO),
1143 1          03 dcr     bit(1)                init(NO),
1144 1          03 scr     bit(1)                init(NO),
1145 1          03 alc     bit(1)                init(NO),
1146 1          03 mfs     bit(1)                init(NO),
1147 1          03 byn     bit(1)                init(NO),
1148 1          03 scl     bit(1)                init(NO),
1149 1          03 fil     bit(2)                init('0'B);
1150 1

```

```

1151 | 1 /*
1152 | 1 /*++
1153 | 1 /*
1154 | 1 /* FUNCTIONAL DESCRIPTION:
1155 | 1 /*
1156 | 1 /*     MONMAIN
1157 | 1 /*
1158 | 1 /*     This routine is the main Monitor routine, entered from DCL.
1159 | 1 /*     It calls the CLE (Command Language Editor) to parse the
1160 | 1 /*     MONITOR command line, and creates a Monitor Request Block (MRB)
1161 | 1 /*     describing the request. The EXECUTE_REQUEST routine is then
1162 | 1 /*     called to execute the request.
1163 | 1 /*
1164 | 1 /* INPUTS:
1165 | 1 /*
1166 | 1 /*     None
1167 | 1 /*
1168 | 1 /* IMPLICIT INPUTS:
1169 | 1 /*
1170 | 1 /*     The MONITOR command line.
1171 | 1 /*
1172 | 1 /* OUTPUTS:
1173 | 1 /*
1174 | 1 /*     None
1175 | 1 /*
1176 | 1 /* ROUTINE VALUE:
1177 | 1 /*
1178 | 1 /*     $$$_NORMAL, or failing MONITOR status.
1179 | 1 /*
1180 | 1 /*--
1181 | 1 /*/
1182 | 1

```

```

1183 1      ON FINISH;                                /* On finish, do nothing */
1184 1      ON ANYCONDITION                            /* On any condition signaled, */
1185 1      BEGIN;
1186 2      DECLARE MON_CODE    FIXED BINARY(31),      /* Monitor message code */
1187 2      TEMP                FIXED BINARY(31),      /* Temporary scratch area */
1188 2      ON_FILE              CHAR(100) VARYING,     /* Holds possible file name string */
1189 2      MNRS_FACNO           FIXED BINARY(31) GLOBALREF VALUE, /* MONITOR facility number */
1190 2      SIGNED_ERR_ENTRY (ANY VALUE, ANY VALUE, ANY VALUE, ANY); /* Rtn to set up PUTMSGVEC */
1191 2
1192 2      CHF$ARGPTR = ONARGSLIST();                  /* Get signal array pointer */
1193 2      ST$VALUE = CHF$SIG_NAME;                     /* Get code for signaled condition */
1194 2      UNSPEC(TEMP) = ST$FAC_NO;                    /* Convert facility no. to binary in TEMP */
1195 2      IF TEMP = MNRS_FACNO                          /* If a MONITOR code, re-signal it */
1196 2      THEN CALL RESIGNAL();
1197 2
1198 2      IF CURR_ERRCODE = MNRS_ERRPARSE &            /* If expecting a CLU parsing error, */
1199 2      CHF$SIG_NAME = CLIS_NOCOMD                    /* AND it's a "No Command on Line", */
1200 2      THEN CALL = NORMAL;                          /* then set up code so it's ignored */
1201 2      ELSE DO;                                     /* Otherwise */
1202 3      IF CURR_ERRCODE = MNRS_ERREXEREA              /* Was it an execute command file read error? */
1203 3      THEN DO;
1204 4      CLOSE FILE (COMMAND_FILE);                  /* Yes, close the execute command file */
1205 4      EXECUTE = NO;                                /* Don't do any more input from the file */
1206 4      END;
1207 3      IF CURR_ERRCODE = 0                          /* See if an error is currently expected */
1208 3      THEN MON_CODE = MNRS_UNEXPERR;                /* No, set "unexpected" code */
1209 3      ELSE MON_CODE = CURR_ERRCODE;                 /* Yes, set currently expected code */
1210 3      CALL SIGNED_ERR(MON_CODE,ST$VALUE,DIM(CHF$SIG_ARG,1),CHF$SIG_ARG); /* Log the error */
1211 3      CALL = MON_CODE;                             /* Set up code for MONITOR request termination */
1212 3      END;
1213 2
1214 2      CURR_ERRCODE = 0;                             /* Reset to default MONITOR error code ('unexpected') */
1215 2      GO TO MON_REQ_TERM;                          /* ... and go terminate (PL/I does an UNWIND) */
1216 2
1217 1      END;

```

```
1218 1 CALL = MONITOR_INIT(); /* Do image-wide initialization */
1219 1 IF STATUS /* Continue if status OK */
1220 1 THEN CALL = MONITOR_CMD(); /* Analyze and execute first (DCL-level) MONITOR cmd */
1221 1
1222 1 MON_REQ_TERM: /* MONITOR request termination */
1223 1
1224 1 /*
1225 1 /* We get to this point by one of three routes:
1226 1 /*
1227 1 /* 1) A MONITOR request has just terminated successfully or with an error status code; or
1228 1 /* 2) A MONITOR request has just terminated with an error that was signaled; or
1229 1 /* 3) The MONITOR_INIT call above terminated with an error status.
1230 1 /*
1231 1 /* In all three cases we want to do the same thing. That is, to loop prompting for more subcommands
1232 1 /* as long as the PROMPT indicator is still set to YES. It can be set to NO by an EXIT subcommand,
1233 1 /* or as a result of the user's striking CTRL/Z (either in response to the MONITOR> prompt, or while
1234 1 /* a MONITOR request is running). In case 3 above, it will always be set to NO. For all cases, the
1235 1 /* variable CALL contains the status code of interest and, if an error, the PUTMSG vector (PUTMSGVEC)
1236 1 /* has been set up with error message information. STATUS is a synonym for the low-order bit of CALL.
1237 1 /*
1238 1 /* If the EXECUTE indicator is set to YES then NEXT_EXECUTE_COMMAND will be called. If EXECUTE is set
1239 1 /* to NO, then NEXT_COMMAND is called. NEXT_EXECUTE_COMMAND will retrieve commands from a file instead
1240 1 /* of the terminal as done by EXECUTE_COMMAND.
1241 1 /*/
1242 1
1243 1 IF STATUS = NOT_SUCCESSFUL /* If bad status, */
1244 1 THEN DO:
1245 2 IF DISPLAYING = YES /* If display output is active, */
1246 2 THEN ST$VALUE = DISPLAY_CLEANUP(); /* then perform cleanup */
1247 2 CALL SIGNAL_MON_ERR(); /* Signal MONITOR error */
1248 2 END;
1249 1
1250 1 DO WHILE (PROMPT = YES); /* Main loop to perform subcommands */
1251 2 IF EXECUTE = YES /* Read from the execute command file? */
1252 2 THEN CALL = NEXT_EXECUTE_COMMAND(); /* Yes, execute next subcommand line from the file */
1253 2 ELSE CALL = NEXT_COMMAND(); /* No, Read from the terminal and execute next subcommand li
1254 2 IF STATUS = NOT_SUCCESSFUL THEN DO: /* If bad status, */
1255 3 IF DISPLAYING = YES /* If display output is active, */
1256 3 THEN ST$VALUE = DISPLAY_CLEANUP(); /* then perform cleanup */
1257 3 CALL SIGNAL_MON_ERR(); /* Signal MONITOR error, using PUTMSGVEC */
1258 3 END;
1259 2 END; /* End of subcommand loop */
1260 1
1261 1
1262 1 ST$VALUE = CALL; /* Get MONITOR completion status */
1263 1 ST$INHIB MSG = YES; /* Inhibit DCL print */
1264 1 RETURN(ST$VALUE); /* Return to DCL */
1265 1
```

1266	1	MONITOR_INIT: Procedure Returns(Fixed Binary(31));
1267	2	
1268	2	/*
1269	2	/* This routine performs general set-up, including
1270	2	/* setting of the current MRB to default values.
1271	2	/*
1272	2	
1273	2	

```
1274 2
1275 2
1276 2
1277 2
1278 2
1279 2
1280 2
1281 2
1282 2
1283 2
1284 2
1285 2
1286 2
1287 2
1288 2
1289 2
1290 2
1291 2
1292 2
1293 2
1294 2
1295 2
1296 2
1297 2
1298 2
1299 2
1300 2
1301 2
1302 2
1303 2
1304 2
1305 2
1306 2
1307 2
1308 2
1309 2
1310 2
1311 2
1312 2
1313 2
1314 2
1315 2

PROMPT = NO;
EXECUTE = NO;
FIRST_MON_CMD = YES;
NORMAC = $$$_NORMAL;

/*
/*      Allocate the two blocks which contain string descriptors for
/*      command qualifiers and default qualifier values. Global
/*      pointers to the blocks are automatically established.
*/

ALLOCATE QUALIFIER_DESC;
ALLOCATE DEF_DESC;

/*
/*      Initialize string descriptors for command
/*      qualifiers, command parameters, etc.
*/

QUAL$$_BEG = LENGTH(BEG_QUAL_S);
QUAL$$_BEG = ADDR(BEG_QUAL_S);
/* Init length longword of descr */
/* Init address longword of descr */

QUAL$$_END = LENGTH(END_QUAL_S);
QUAL$$_END = ADDR(END_QUAL_S);
/* Init length longword of descr */
/* Init address longword of descr */

QUAL$$_INT = LENGTH(INT_QUAL_S);
QUAL$$_INT = ADDR(INT_QUAL_S);
/* Init length longword of descr */
/* Init address longword of descr */

QUAL$$_FLUSH = LENGTH(FLUSH_QUAL_S);
QUAL$$_FLUSH = ADDR(FLUSH_QUAL_S);
/* Init length longword of descr */
/* Init address longword of descr */

QUAL$$_VIEW = LENGTH(VIEW_QUAL_S);
QUAL$$_VIEW = ADDR(VIEW_QUAL_S);
/* Init length longword of descr */
/* Init address longword of descr */

QUAL$$_BY_NODE = LENGTH(BY_NODE_QUAL_S);
QUAL$$_BY_NODE = ADDR(BY_NODE_QUAL_S);
/* Init length longword of descr */
/* Init address longword of descr */

QUAL$$_INP = LENGTH(INP_QUAL_S);
QUAL$$_INP = ADDR(INP_QUAL_S);
/* Init length longword of descr */
/* Init address longword of descr */
```

```

1316 QUAL$DISP = LENGTH(DISP_QUAL_S); /* Init length longword of descr */
1317 QUAL$A_DISP = ADDR(DISP_QUAL_ST); /* Init address longword of descr */
1318
1319 QUAL$REC = LENGTH(REC_QUAL_S); /* Init length longword of descr */
1320 QUAL$A_REC = ADDR(REC_QUAL_ST); /* Init address longword of descr */
1321
1322 QUAL$SUMM = LENGTH(SUMM_QUAL_S); /* Init length longword of descr */
1323 QUAL$A_SUMM = ADDR(SUMM_QUAL_ST); /* Init address longword of descr */
1324
1325 QUAL$COMM = LENGTH(COMM_QUAL_S); /* Init length longword of descr */
1326 QUAL$A_COMM = ADDR(COMM_QUAL_ST); /* Init address longword of descr */
1327
1328 QUAL$BY_NODE = LENGTH(BY_NODE_QUAL_S); /* Init length longword of descr */
1329 QUAL$A_BY_NODE = ADDR(BY_NODE_QUAL_ST); /* Init address longword of descr */
1330
1331 QUAL$CLASS = LENGTH(CLASS_PARM_S); /* Init length longword of descr */
1332 QUAL$A_CLASS = ADDR(CLASS_PARM_ST); /* Init address longword of descr */
1333
1334 QUAL$ALL = LENGTH(ALL_QUAL_S); /* Init length longword of descr */
1335 QUAL$A_ALL = ADDR(ALL_QUAL_ST); /* Init address longword of descr */
1336
1337 QUAL$CUR = LENGTH(CUR_QUAL_S); /* Init length longword of descr */
1338 QUAL$A_CUR = ADDR(CUR_QUAL_ST); /* Init address longword of descr */
1339
1340 QUAL$AVE = LENGTH(AVE_QUAL_S); /* Init length longword of descr */
1341 QUAL$A_AVE = ADDR(AVE_QUAL_ST); /* Init address longword of descr */
1342
1343 QUAL$MIN = LENGTH(MIN_QUAL_S); /* Init length longword of descr */
1344 QUAL$A_MIN = ADDR(MIN_QUAL_ST); /* Init address longword of descr */
1345
1346 QUAL$MAX = LENGTH(MAX_QUAL_S); /* Init length longword of descr */
1347 QUAL$A_MAX = ADDR(MAX_QUAL_ST); /* Init address longword of descr */
1348
1349 QUAL$TOPC = LENGTH(TOPC_QUAL_S); /* Init length longword of descr */
1350 QUAL$A_TOPC = ADDR(TOPC_QUAL_ST); /* Init address longword of descr */
1351
1352 QUAL$TOPD = LENGTH(TOPD_QUAL_S); /* Init length longword of descr */
1353 QUAL$A_TOPD = ADDR(TOPD_QUAL_ST); /* Init address longword of descr */
1354
1355 QUAL$TOPB = LENGTH(TOPB_QUAL_S); /* Init length longword of descr */
1356 QUAL$A_TOPB = ADDR(TOPB_QUAL_ST); /* Init address longword of descr */
1357
1358 QUAL$TOPF = LENGTH(TOPF_QUAL_S); /* Init length longword of descr */
1359 QUAL$A_TOPF = ADDR(TOPF_QUAL_ST); /* Init address longword of descr */
1360
1361 QUAL$CPU = LENGTH(CPU_QUAL_S); /* Init length longword of descr */
1362 QUAL$A_CPU = ADDR(CPU_QUAL_ST); /* Init address longword of descr */
1363
1364 QUAL$PCENT = LENGTH(PCENT_QUAL_S); /* Init length longword of descr */
1365 QUAL$A_PCENT = ADDR(PCENT_QUAL_ST); /* Init address longword of descr */
1366
1367 QUAL$ITEM = LENGTH(ITEM_QUAL_S); /* Init length longword of descr */
1368 QUAL$A_ITEM = ADDR(ITEM_QUAL_ST); /* Init address longword of descr */
1369
1370 DEF$REC = LENGTH(REC_DEF_S); /* Init length longword of descr */
1371 DEF$A_REC = ADDR(REC_DEF_ST); /* Init address longword of descr */

```

MONMAIN
V04-000

E 12
16-SEP-1984 02:10:57 VAX-11 PL/I X2.1-273 Page 19
5-SEP-1984 15:09:57 ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (16)

1372 2
1373 2
1374 2
1375 2
1376 2
1377 2
1378 2
1379 2

DEF\$L_DISP = LENGTH(DISP_DEF_S);
DEF\$A_DISP = ADDR(DISP_DEF_S);

DEF\$L_SUMM = LENGTH(SUMM_DEF_S);
DEF\$A_SUMM = ADDR(SUMM_DEF_S);

/* Init length longword of descr */
/* Init address longword of descr */

/* Init length longword of descr */
/* Init address longword of descr */

```

1380 : /*
1381 : /*      Initialize STAT_TABLE descriptor pointers.
1382 : */
1383 :
1384 : STAT_DESC(ALL_STAT) = ADDR(QUAL$$_ALL);          /* Init pointer to ALL descr */
1385 : STAT_DESC(CUR_STAT) = ADDR(QUAL$$_CUR);          /* Init pointer to CUR descr */
1386 : STAT_DESC(AVE_STAT) = ADDR(QUAL$$_AVE);          /* Init pointer to AVE descr */
1387 : STAT_DESC(MIN_STAT) = ADDR(QUAL$$_MIN);          /* Init pointer to MIN descr */
1388 : STAT_DESC(MAX_STAT) = ADDR(QUAL$$_MAX);          /* Init pointer to MAX descr */
1389 :
1390 : /*
1391 : /*      Initialize PROCD_TABLE descriptor pointers.
1392 : */
1393 :
1394 : PROCD_DESC(REG_PROC) = NULL();                  /* Indicate no qualifier for "regular" PROCESSES display */
1395 : PROCD_DESC(TOPC_PROC) = ADDR(QUAL$$_TOPC);        /* Init pointer to TOPC descr */
1396 : PROCD_DESC(TOPD_PROC) = ADDR(QUAL$$_TOPD);        /* Init pointer to TOPD descr */
1397 : PROCD_DESC(TOPB_PROC) = ADDR(QUAL$$_TOPB);        /* Init pointer to TOPB descr */
1398 : PROCD_DESC(TOPF_PROC) = ADDR(QUAL$$_TOPF);        /* Init pointer to TOPF descr */
1399 :
1400 : MRB_DEF.DIS = ADDR(DEF$$_DISP);                  /* Init default value of ptr to display file descr */
1401 : ALLOCATE MCA SET(MCAPTR);                          /* Obtain Monitor Communication Area */
1402 : ALLOCATE SYS_INFO SET(SPTR);                       /* Obtain System Information Area */
1403 :
1404 : /*
1405 : /*      Allocate "temp", "current" and "active" MRBs. Another MRB (MRB DEF),
1406 : /*      the "default" MRB, is allocated in the data section above. Each MRB fully describes
1407 : /*      a MONITOR request. The usage of the MRBs is analogous to that of SYSGEN PAR files.
1408 : /*      To wit, the "default" MRB never changes; it is moved to the "current" MRB whenever
1409 : /*      an INITIALIZE subcommand is issued; an implicit INITIALIZE occurs when the MONITOR
1410 : /*      image is invoked. The "current" MRB may be changed repeatedly with SET commands without
1411 : /*      actually executing a request; When a MONITOR request is executed (with a MONITOR
1412 : /*      subcommand, or a MONITOR DCL command), the "current" MRB is moved to the "active" MRB,
1413 : /*      the "active" MRB is updated in accordance with the MONITOR request, and the request
1414 : /*      is executed using the "active" MRB. The "temp" MRB is used to accumulate information
1415 : /*      resulting from a SET command. If the command contains no errors, "temp" is swapped
1416 : /*      with "current"; otherwise, "temp" is discarded. This is done to avoid defining a partial
1417 : /*      request when the SET command contains an error.
1418 : */
1419 :
1420 : ALLOCATE MRB SET(TEMP_MRBPTR);                    /* Allocate the "temp" MRB */
1421 : ALLOCATE MRB SET(CURR_MRBPTR);                    /* Allocate the "current" MRB */
1422 : ALLOCATE MRB SET(ACT_MRBPTR);                     /* Allocate the "active" MRB */
1423 : DEF_MRBPTR = ADDR(MRB_DEF);                       /* Init pointer to "default" MRB */
1424 :
1425 : CALL = INIT_CMD();                                /* Do an INITIALIZE command ... i.e., move default */
1426 : /* ... MRB to current MRB */
1427 :
1428 : RETURN(CALL);                                     /* Return with status from INIT_CMD */
1429 : END MONITOR_INIT;
1430 :

```

```
1431 1 NEXT_COMMAND: Procedure Returns(Fixed Binary(31)); /* Routine to read the next subcommand & execute it */
1432 2
1433 3 /*++
1434 4 /*
1435 5 /* FUNCTIONAL DESCRIPTION:
1436 6 /*
1437 7 /* NEXT_COMMAND
1438 8 /*
1439 9 /* This routine is called by the main routine to read the next subcommand and execute it.
1440 10 /* It returns a status code indicating the disposition of the command.
1441 11 /*
1442 12 /* INPUTS:
1443 13 /*
1444 14 /* None
1445 15 /*
1446 16 /* IMPLICIT INPUTS:
1447 17 /*
1448 18 /* PROMPT -- a bit indicating whether or not to prompt for a command.
1449 19 /* It has always been set to YES before entry to NEXT_COMMAND.
1450 20 /*
1451 21 /* OUTPUTS:
1452 22 /*
1453 23 /* None
1454 24 /*
1455 25 /* IMPLICIT OUTPUTS:
1456 26 /*
1457 27 /* The next command is executed.
1458 28 /*
1459 29 /* PROMPT -- a bit indicating whether or not to prompt for another
1460 30 /* command. It is always set to YES before entry to NEXT_COMMAND, and
1461 31 /* can be set to NO by NEXT_COMMAND in any of the following situations:
1462 32 /*
1463 33 /* 1) The user issues the EXIT subcommand.
1464 34 /* 2) The user strikes CTRL/Z in response to the MONITOR> prompt.
1465 35 /* 3) The user strikes CTRL/Z while a MONITOR subcommand is running.
1466 36 /*
1467 37 /* PUTMSGVEC -- a 20-longword vector which is loaded with error message
1468 38 /* information by NEXT_COMMAND whenever an error status is
1469 39 /* returned to the caller of NEXT_COMMAND.
1470 40 /*
1471 41 /* ROUTINE VALUE:
1472 42 /*
1473 43 /* A status code indicating the disposition of the command. If an
1474 44 /* error status, then the PUTMSGVEC error message vector will have
1475 45 /* been set up.
1476 46 /*
1477 47 /* SIDE EFFECTS:
1478 48 /*
1479 49 /* None
1480 50 /*
1481 51 /*/
1482 52
```



```
1539      2 L      FIXED BINARY(31),          /* Length */
1540      2 A      POINTER,                    /* Address */
1541
1542      1 REP_LINE,                          /* Static command line descriptor (for subcommands)
1543      2 L      FIXED BINARY(31),          /* Length */
1544      2 A      POINTER,                    /* Address */
1545
1546      CMD_LINE_S  CHAR(MAX_EXEC_LINE) STATIC; /* Command buffer to replace '@' with 'EXECUTE ' */
1547
1548
1549      Declare
1550      MONSUB      CHAR(1) GLOBALREF;        /* Command language definition tables ... */
1551
1552
1553      Declare
1554      AT_SIGN_S   CHAR(1) STATIC INIT('@'), /* Note -- we simply need a global reference to the
1555      EXECUTE_S   CHAR(8) STATIC INIT('EXECUTE '), /* tables. Their length is unknown and irrelevant */
1556      AT_SIGN_POS FIXED BINARY(31);        /* '@' used to search command line */
1557
1558
1559      FIRST_MON_CMD = NO;                  /* 'EXECUTE ' used to replace the '@' in the command
1560      DYN_STRING.L = 0;                    /* Position of '@' in the command line */
1561
1562
1563      DO WHILE (DYN_STRING.L = 0);          /* First MONITOR cmd executes before NEXT_COMMAND rt
1564      CALL = LIB$GET_INPUT(DYN_STRING,PROMPT_STR,); /* Init cmd line length to enter loop */
1565      IF STATUS = NOT_SUCCESSFUL           /* Loop while user enters null lines */
1566      THEN DO;                             /* Read the next subcommand */
1567          PROMPT = NO;                     /* If LIB$GET_INPUT call failed, */
1568          IF CALL = RMSS$ EOF              /* Indicate no more prompting */
1569          THEN RETURN(NORMAL);             /* If end-of-input, */
1570          ELSE DO;                         /* then return with normal status */
1571              CALL MON_ERR(MNRS$ ERRPROMPT, CALL); /* Otherwise, log the error ... */
1572              RETURN(MNRS$ ERRPROMPT);          /* and return with status */
1573          END;
1574      END;
1575
1576      CMD_LINE.L = DYN_STRING.L;           /* Copy the length of the command line */
1577      CMD_LINE.A = ADDR(CMD_LINE_S);       /* Get the address of the new working buffer */
1578      CMD_LINE_S = DYN_STRING_S;           /* Copy the command line into the buffer */
1579      AT_SIGN_POS = STR$POSITION(CMD_LINE, DESCRIPTOR(AT_SIGN_S)); /* Locate a '@' in the command line */
1580      IF AT_SIGN_POS > 0                   /* Was there one? */
1581      THEN DO;                             /* Yes, prepare to replace the '@' with 'EXECUTE ' */
1582          REP_LINE.L = CMD_LINE.L + 7;     /* Add space for 'EXECUTE ' in replacement desc */
1583          REP_LINE.A = CMD_LINE.A;         /* Get address of replacement string */
1584          CALL = STR$REPLACE (REP_LINE, CMD_LINE, AT_SIGN_POS, /* Replace the '@' with 'EXECUTE ' */
1585                             AT_SIGN_POS, DESCRIPTOR(EXECUTE_S)); /* (need REP_LINE to prevent trunc warning) */
1586          IF STATUS = NOT_SUCCESSFUL        /* If STR$REPLACE call failed, */
1587          THEN DO;
1588              CALL MON_ERR(MNRS$ ERREXEREP, CALL); /* Log the error ... */
1589              RETURN(MNRS$ ERREXEREP);        /* and return with status */
1590          END;
1591          ELSE CMD_LINE.L = REP_LINE.L;     /* STR$REPLACE succeeded, update length of descr */
1592      END;
1593
```

1594	2	CURR_ERRCODE = MNRS_ERRPARSE;	/* Set MONITOR code in case parsing error signaled *
1595	2	CALL = CLISDCL_PARSE(CMD_LINE,MONSUB);	/* Parse the subcommand */
1596	2	CURR_ERRCODE = 0;	/* Reset to default MONITOR code */
1597	2	IF STATUS = NOT_SUCCESSFUL	/* If parse failed, */
1598	2	THEN DO;	
1599	2	IF CALL = CLIS_NOCOMD	/* If 'no command on line' */
1600	2	THEN RETURN(NORMAL);	/* then quietly ignore it */
1601	3	ELSE DO;	/* Otherwise, */
1602	4	CALL MON_ERR(MNRS_ERRPARSE,CALL);	/* log the error ... */
1603	4	RETURN(MNRS_ERRPARSE);	/* ... and return with status */
1604	4	END;	
1605	2	END;	
1606	2		
1607	2	CALL = CLISDISPATCH();	/* Execute the parsed command */
1608	2		/* Note -- command subroutines return status */
1609	2		/* values and log their own errors by */
1610	2		/* calling MON_ERR */
1611	2		
1612	2	CURR_ERRCODE = 0;	/* Reset to default MONITOR code in case subcommand
1613	2	RETURN(CALL);	/* Return to caller with cmd subroutine's status */
1614	2		
1615	2	END NEXT_COMMAND;	
1616	1		
1617	1		

```
1618 1 NEXT_EXECUTE_COMMAND: Procedure Returns(Fixed Binary(31)); /* Routine to read the next subcommand & execute it
1619 2
1620 3
1621 4
1622 5
1623 6
1624 7
1625 8
1626 9
1627 10
1628 11
1629 12
1630 13
1631 14
1632 15
1633 16
1634 17
1635 18
1636 19
1637 20
1638 21
1639 22
1640 23
1641 24
1642 25
1643 26
1644 27
1645 28
1646 29
1647 30
1648 31
1649 32
1650 33
1651 34
1652 35
1653 36
1654 37
1655 38
1656 39
1657 40
1658 41
1659 42
1660 43
1661 44
1662 45
1663 46
1664 47
1665 48
1666 49
1667 50
1668 51
1669 52
1670 53
1671 54
1672 55
1673 56
```

```
/*+++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/* NEXT_EXECUTE_COMMAND
/*
/* This routine is called by the main routine to read the next subcommand from
/* the execute command file and execute it. It returns a status code indicating
/* the disposition of the command.
/*
/* INPUTS:
/*
/* None
/*
/* IMPLICIT INPUTS:
/*
/* COMMAND_FILE - file reference used to read the execute command file.
/*
/* EXECUTE - a bit indicating whether or not commands are read from an execute
/* command file. It has been set to YES before entry to NEXT_EXECUTE_COMMAND.
/*
/* OUTPUTS:
/*
/* None
/*
/* IMPLICIT OUTPUTS:
/*
/* The next command is executed.
/*
/* EXECUTE - a bit indicating whether or not to read from the execute command file
/* for another command. It is always set to YES before entry to NEXT_EXECUTE_COMMAND,
/* and can be set to NO by NEXT_EXECUTE_COMMAND in any of the following situations:
/*
/* 1) Execute command file EOF is encountered.
/* 2) The user issues the EXIT subcommand.
/* 3) The user strikes CTRL/Z in response to the MONITOR> prompt.
/* 4) The user strikes CTRL/Z while a MONITOR subcommand is running.
/* 5) The file cannot be opened properly.
/* 6) An error occurs when trying to read the file.
/*
/* PUTMSGVEC -- a 20-longword vector which is loaded with error message
/* information by NEXT_EXECUTE_COMMAND whenever an error status is
/* returned to the caller of NEXT_EXECUTE_COMMAND.
/*
/* ROUTINE VALUE:
/*
/* A status code indicating the disposition of the command. If an error status,
/* then the PUTMSGVEC error message vector will have been set up.
/*
/* MNRS_ERRPARSE - error parsing command from the execute command file.
/* MNRS_ERREXEREA - error reading execute command file.
/* SSS_NORMAL - Success.
/*
/* SIDE EFFECTS:
```

MONMAIN
V04-000

L 12
16-SEP-1984 02:11:02
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (21)

Page 26

1674	:	2	/*	None.
1675	:	2	/*	
1676	:	2	/*	
1677	:	2	/*	
1678	:	2	/*	

```
1679 | 2 | /*
1680 | 2 | /*
1681 | 2 | /*
1682 | 2 | /*
1683 | 2 | /*
1684 | 2 | /*
1685 | 2 | /*
1686 | 2 | /*
1687 | 2 | %REPLACE          MAX_EXEC_LINE          BY 256;          /* Max execute file line size */
1688 | 2 |
1689 | 2 | /*
1690 | 2 | /*
1691 | 2 | /*
1692 | 2 | /*
1693 | 2 | /*
1694 | 2 | /*
1695 | 2 | /*
1696 | 2 | /*
1697 | 2 | Declare
1698 | 2 |     CLISDCL_PARSE      EXTERNAL ENTRY(ANY, ANY)          /* Rtn to do DCL-like parsing of cmd line */
1699 | 2 |     RETURNS(FIXED BINARY(31)),
1700 | 2 |     CLISDISPATCH      EXTERNAL ENTRY
1701 | 2 |     RETURNS(FIXED BINARY(31)),          /* Rtn to dispatch to latest parsed cmd */
1702 | 2 |     CLISGET_VALUE      ENTRY(ANY, ANY)
1703 | 2 |     RETURNS(FIXED BINARY(31)),          /* CLE routine to get qualifier values */
1704 | 2 |     STR$POSITION      EXTERNAL ENTRY(ANY, ANY, ANY)
1705 | 2 |     OPTIONS(VARIABLE)
1706 | 2 |     RETURNS(FIXED BINARY(31)),          /* Rtn to locate 1st occurrence of a substring */
1707 | 2 |     STR$REPLACE      EXTERNAL ENTRY(ANY, ANY, ANY, ANY, ANY) /* Rtn to replace a substring with another substring
1708 | 2 |     RETURNS(FIXED BINARY(31));
1709 | 2 |
1710 | 2 | Declare
1711 | 2 |     COMMAND_FILE      FILE          GLOBALREF,
1712 | 2 |     MONSUB            CHAR(1)       GLOBALREF,          /* Execute Command File */
1713 | 2 |
1714 | 2 |     NORMAL            FIXED BINARY(31) GLOBALREF,
1715 | 2 |     EXECUTE           BIT(1) ALIGNED GLOBALREF;          /* Command language definition tables ... */
1716 | 2 |
1717 | 2 |     /* Note -- we simply need a global reference to the
1718 | 2 |     /* tables. Their length is unknown and irrelevant */
1719 | 2 |     /* MONITOR normal return status */
1720 | 2 |     /* YES => read another command from the execute file
1721 | 2 |
1722 | 2 | Declare
1723 | 2 |     MNRS_ERREXEREP    FIXED BINARY(31) GLOBALREF VALUE,          /* Error message code */
1724 | 2 |     MNRS_ERREXEREA    FIXED BINARY(31) GLOBALREF VALUE,          /* Error message code */
1725 | 2 |     MNRS_ERRPARSE     FIXED BINARY(31) GLOBALREF VALUE;          /* Error message code */
1726 | 2 |
1727 | 2 | /*
1728 | 2 | /*
1729 | 2 | /*
1730 | 2 | /*
1731 | 2 | /*
1732 | 2 | /*
1733 | 2 | Declare
1734 | 2 |     1 SUB_COMMAND     GLOBALDEF,
```

```
1735      2 L      FIXED BINARY(31),
1736      2 A      POINTER,
1737      COMMAND_RECORD CHAR(MAX_EXEC_LINE) STATIC VARYING GLOBALDEF; /* Buffer for a subcommand */
1738
1739
1740      /*
1741      /*
1742      /*
1743      /*
1744      /*
1745      /*
1746      /*/
1747
1748
1749      Declare
1750      1 REP_LINE,
1751      2 L      FIXED BINARY(31),
1752      2 A      POINTER;
1753
1754      Declare
1755      AT_SIGN_S CHAR(1) STATIC INIT('a'),
1756      EXECUTE_S CHAR(8) STATIC INIT('EXECUTE '),
1757      AT_SIGN_POS FIXED BINARY(31),
1758      TEMP_COMMAND_PTR FIXED BINARY(31)
1759      BASED(ADDR(SUB_COMMAND.A));
1760
1761
1762      ON ENDFILE (COMMAND_FILE) GOTO COMMAND_EOF;
1763
1764
1765      CURR_ERRCODE = MNRS_ERREXEREA;
1766      READ_FILE (COMMAND_FILE) INTO (COMMAND_RECORD);
1767      CURR_ERRCODE = 0;
1768
1769      SUB_COMMAND.L = LENGTH(COMMAND_RECORD);
1770      SUB_COMMAND.A = ADDR(COMMAND_RECORD);
1771      TEMP_COMMAND_PTR = TEMP_COMMAND_PTR + 2;
1772      AT_SIGN_POS = STR$POSITION(SUB_COMMAND, DESCRIPTOR(AT_SIGN_S));
1773      IF AT_SIGN_POS > 0
1774      THEN DO;
1775      REP_LINE.L = SUB_COMMAND.L + 7;
1776      REP_LINE.A = SUB_COMMAND.A;
1777      CALL = STR$REPLACE (REP_LINE, SUB_COMMAND, AT_SIGN_POS,
1778      AT_SIGN_POS, DESCRIPTOR(EXECUTE_S));
1779      IF STATUS = NOT_SUCCESSFUL
1780      THEN DO;
1781      CALL MON_ERR(MNRS_ERREXEREP,CALL);
1782      RETURN(MNRS_ERREXEREP);
1783      END;
1784      ELSE SUB_COMMAND.L = REP_LINE.L;
1785      END;
1786
```

LOCAL STORAGE

/* Static command line descriptor (for subcommands)
/* Length */
/* Address */

/* 'a' used to search command line */
/* 'EXECUTE ' used to replace 'a' */
/* Position of 'a' in command line */
/* Alias for SUB_COMMAND.A computation */

/* Set up the EOF condition */

/* Set MONITOR code in case read error is signaled */
/* Read the next subcommand */
/* Reset the error code, assume the condition was re

/* Set the length of the SUB_COMMAND descriptor */
/* Set the address of the SUB_COMMAND descriptor */
/* Advance ptr beyond length word */
/* Locate 'a' in command line */
/* Was there a 'a'? */
/* Yes, prepare to replace 'a' with 'EXECUTE ' */
/* Add space for 'EXECUTE ' in replacement string */
/* Get address of subcommand buffer */
/* Replace 'a' with 'EXECUTE ' */
/* (REP_LINE prevents truncation warning) */
/* If STR\$REPLACE call failed, */

/* Log the error ... */
/* and return with status */

/* STR\$REPLACE succeeded, upade command desc */

```
1787 2  CURR_ERRCODE = MNRS$ ERRPARSE;  
1788 2  CALL = CLISDCL_PARSE(SUB_COMMAND,MONSUB);  
1789 2  CURR_ERRCODE = 0;  
1790 2  IF STATUS = NOT_SUCCESSFUL  
1791 2  THEN DO;  
1792 3  IF CALL = CLIS$ NOCMD  
1793 3  THEN RETURN(NORMAL);  
1794 3  ELSE DO;  
1795 4  CALL MON_ERR(MNRS$ ERRPARSE,CALL);  
1796 4  RETURN(MNRS$ ERRPARSE);  
1797 4  END;  
1798 3  END;  
1799 2  CALL = CLIS$DISPATCH();  
1800 2  
1801 2  
1802 2  
1803 2  
1804 2  CURR_ERRCODE = 0;  
1805 2  RETURN(CALL);  
1806 2  
1807 2  
1808 2  COMMAND_EOF:  
1809 2  CLOSE FILE (COMMAND_FILE);  
1810 2  EXECUTE = NO;  
1811 2  CURR_ERRCODE = 0;  
1812 2  RETURN(NORMAL);  
1813 2  
1814 2  END NEXT_EXECUTE_COMMAND;  
1815 1  
1816 1  END MONMAIN;  
1817
```

```
/* Set MONITOR code in case parsing error signaled */  
/* Parse the subcommand */  
/* Reset to default MONITOR code */  
/* If parse failed, */  
  
/* If "no command on line" */  
/* then quietly ignore it */  
/* Otherwise, */  
/* log the error ... */  
/* ... and return with status */  
  
/* Execute the parsed command */  
/* Note -- command subroutines return status */  
/* values and log their own errors by */  
/* calling MON_ERR */  
/* Reset to default MONITOR code in case subcommand */  
/* Return to caller with cmd subroutine's status */  
  
/* Close the file after EOF condition raised */  
/* Indicate no more from the execute file */  
/* Reset to default MONITOR code in case subcommand */  
/* Return to caller with cmd subroutine's status */
```

```

1818 GET_QUALIFIERS: Procedure Returns(Fixed Binary(31));
1819
1820 /*++
1821 /*
1822 /* FUNCTIONAL DESCRIPTION:
1823 /*
1824 /*     GET_QUALIFIERS
1825 /*
1826 /*     Communicate with CLE to get qualifier settings and
1827 /*     their values. Record all such info in the MRB (Monitor
1828 /*     Request Block) pointed to by the current value of MRBPTR
1829 /*     (it may be the "current" or the "active" MRB).
1830 /*
1831 /* INPUTS:
1832 /*
1833 /*     TBS
1834 /*
1835 /* IMPLICIT INPUTS:
1836 /*
1837 /*     TBS
1838 /*
1839 /* OUTPUTS:
1840 /*
1841 /*     TBS
1842 /*
1843 /* IMPLICIT OUTPUTS:
1844 /*
1845 /*     TBS
1846 /*
1847 /* ROUTINE VALUE:
1848 /*
1849 /*     TBS
1850 /*
1851 /* SIDE EFFECTS:
1852 /*
1853 /*     TBS
1854 /*
1855 /*/
1856

```

```
1857 | 1 /*
1858 | 1 /*
1859 | 1 /*
1860 | 1 /*
1861 | 1 /*
1862 | 1 /*
1863 | 1 /*/
1864 | 1
1865 | 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
2633 | 1
2634 | 1 /*
2635 | 1 /*
2636 | 1 /*
2637 | 1 /*
2638 | 1 /*
2639 | 1 /*
2640 | 1 /*/
2641 | 1
2642 | 1 %INCLUDE SYSS$GETTIM; /* $GETTIM system service */
2648 | 1
2649 | 1 /*
2650 | 1 /*
2651 | 1 /*
2652 | 1 /*
2653 | 1 /*
2654 | 1 /*/
2655 | 1
2656 | 1 Declare
2657 | 1 CLIS$PRESENT FIXED BINARY(31) GLOBALREF VALUE, /* CLIS$PRESENT return status code for 'explicitly pr
2658 | 1 CLIS$NEGATED FIXED BINARY(31) GLOBALREF VALUE, /* CLIS$PRESENT return status code for 'explicitly ne
2659 | 1 NORMAL FIXED BINARY(31) GLOBALREF, /* MONITOR normal return status */
2660 | 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
2661 | 1 M POINTER DEFINED(MRBPTR), /* Short-hand synonym for MRBPTR */
2662 | 1 DEF MRBPTR POINTER GLOBALREF, /* Pointer to 'default' MRB (Monitor Request Block)
2663 | 1 MCAPTR POINTER GLOBALREF, /* Pointer to MCA (Monitor Communication Area) */
2664 | 1 MC POINTER DEFINED(MCAPTR), /* Short-hand synonym for MCAPTR */
2665 | 1 QUALPTR POINTER GLOBALREF, /* Pointer to Qualifier Descriptors Block */
2666 | 1 DEFPTR POINTER GLOBALREF, /* Pointer to Qualifier Default Value Descriptors Bl
2667 | 1 CURR_ERRCODE FIXED BINARY(31) GLOBALREF; /* MONITOR error status code currently expected */
2668 | 1
2669 | 1 Declare
2670 | 1 INP_PTR_VOL POINTER GLOBALREF, /* Pointer to volatile /INPUT file-spec */
2671 | 1 DISP_PTR_VOL POINTER GLOBALREF, /* Pointer to volatile /DISPLAY file-spec */
2672 | 1 REC_PTR_VOL POINTER GLOBALREF, /* Pointer to volatile /RECORD file-spec */
2673 | 1 SUMM_PTR_VOL POINTER GLOBALREF, /* Pointer to volatile /SUMMARY file-spec */
2674 | 1 COMM_PTR_VOL POINTER GLOBALREF; /* Pointer to volatile /COMMENT string */
2675 | 1
2676 | 1 Declare
2677 | 1 INP_PTR_SWAP BIT(1) ALIGNED GLOBALREF, /* YES => swap INP_PTR_VOL and INP_PTR_PERM */
2678 | 1 DISP_PTR_SWAP BIT(1) ALIGNED GLOBALREF, /* YES => swap DISP_PTR_VOL and DISP_PTR_PERM */
2679 | 1 REC_PTR_SWAP BIT(1) ALIGNED GLOBALREF, /* YES => swap REC_PTR_VOL and REC_PTR_PERM */
2680 | 1 SUMM_PTR_SWAP BIT(1) ALIGNED GLOBALREF, /* YES => swap SUMM_PTR_VOL and SUMM_PTR_PERM */
2681 | 1 COMM_PTR_SWAP BIT(1) ALIGNED GLOBALREF; /* YES => swap COMM_PTR_VOL and COMM_PTR_PERM */
2682 | 1
```

```
2683 1 Declare
2684 1 1 DYN_STRING GLOBALREF, /* Dynamic string descriptor */
2685 1 2 L FIXED BINARY(15), /* Length */
2686 1 2 TC CHAR(2), /* Type and Class */
2687 1 2 A POINTER, /* Address */
2688 1
2689 1 DYN_STRING_S CHAR(DYN_STRING.L) BASED(DYN_STRING.A); /* String */
2690 1
2691 1 /*
2692 1 /* -----+
2693 1 /*
2694 1 /* EXTERNAL ROUTINE DEFINITIONS
2695 1 /*
2696 1 /* -----+
2697 1 /*/
2698 1
2699 1 Declare
2700 1 CLISGET_VALUE ENTRY(ANY, ANY, FIXED BINARY(15)) /* CLE routine to get qualifier values */
2701 1 OPTIONS(VARIABLE) RETURNS(BIT(1)),
2702 1 CLISPRESNT EXTERNAL ENTRY(ANY) RETURNS(FIXED BINARY(31)), /* CLE routine to determine presence of qualifiers */
2703 1 LIBSCVT_TIME EXTERNAL ENTRY(ANY, BIT(64) ALIGNED) /* RTL routine to convert to qword time val */
2704 1 RETURNS(BIT(1)),
2705 1 STR$UPCASE EXTERNAL ENTRY(ANY,ANY) RETURNS(BIT(1)), /* RTL routine to upcase a string */
2706 1 MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE); /* MONITOR MACRO-32 routine to log synchronous error
2707 1
2708 1 /*
2709 1 /* -----+
2710 1 /*
2711 1 /* MESSAGE DEFINITIONS
2712 1 /*
2713 1 /* -----+
2714 1 /*/
2715 1
2716 1 Declare
2717 1 MNRS_INVBTIMSP FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
2718 1 MNRS_INVETIMSP FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
2719 1 MNRS_INVINTSP FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
2720 1 MNRS_INVFLUSHSP FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
2721 1 MNRS_INVVIEWSP FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
2722 1 MNRS_INVINPFIL FIXED BINARY(31) GLOBALREF VALUE; /* Error message code */
2723 1 /*
2724 1 /* -----+
2725 1 /*
2726 1 /* GLOBAL STORAGE DEFINITIONS
2727 1 /*
2728 1 /* -----+
2729 1 /*/
2730 1
2731 1 Declare
2732 1 QUAL_SPECIFIED BIT(1) ALIGNED GLOBALDEF INIT('0'B); /* YES => at least 1 qualifier explicitly spec'd */
2733 1
2734 1 /*
2735 1 /* -----+
2736 1 /*
2737 1 /* COMPILE-TIME CONSTANTS
2738 1 /* -----+
2738 1 /*
```

MONMAIN
V04-000

F 13
16-SEP-1984 02:11:07
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (26)

Page 33

```
2739 : 1 /* -----+
2740 : 1 /*/
2741 : 1
2742 1 %REPLACE SECONDS TOK SIZE BY 7: /* Size of token for seconds */
2743 1 %REPLACE TIME_TOK_SIZE BY 40: /* Size of token for time specs */
2744 1 %REPLACE FILE_SPEC_SIZE BY 128: /* Max file spec size */
2745 1
```

```
2746 1 /*
2747 1 /*
2748 1 /*
2749 1 /*
2750 1 /*
2751 1 /*
2752 1 /*
2753 1 /*
2754 1 /*
2755 1 /*
2756 1 /*
2757 1 /*
2758 1 /*
2759 1 /*
2760 1 /*
2761 1
2762 1 Declare
2763 1 TEMP          FIXED BINARY(31),          /* Temporary "scratch" area */
2764 1 CALL          FIXED BINARY(31),          /* Holds function value (return status) of called ro
2765 1 STATUS        BIT(1) BASED(ADDR(CALL));   /* Low-order status bit for called routines */
2766 1
2767 1 Declare
2768 1 FILE_SPEC_PTR POINTER,                    /* Pointer to structure consisting of file-spec ...
2769 1 PARSED_SPEC_PTR POINTER,                  /* ... string descriptor followed by string itself */
2770 1 COMM_STR_PTR  POINTER,                    /* Pointer to structure consisting of file-spec ...
2771 1 COMM_STR_PTR  POINTER;                    /* ... string descriptor followed by string itself */
2772 1
2773 1
2774 1
2775 1 Declare
2776 1 FILE_SPEC     BASED(FILE_SPEC_PTR),       /* File-spec string descriptor and string */
2777 1 2 L           FIXED BINARY(31),           /* File-spec string length */
2778 1 2 A           POINTER,                     /* File-spec string address */
2779 1 2 S           CHAR(FILE_SPEC_SIZE),        /* File-spec string */
2780 1
2781 1
2782 1 PARSED_SPEC   BASED(PARSED_SPEC_PTR),     /* Parsed File-spec dynamic string descriptor and st
2783 1 2 L           FIXED BINARY(15),           /* Length */
2784 1 2 T           FIXED BINARY(7),           /* Type */
2785 1 2 C           FIXED BINARY(7),           /* Class */
2786 1 2 A           POINTER,                     /* Address */
2787 1
2788 1
2789 1 DYN_SPEC      GLOBALDEF,                  /* Dynamic File-spec string descriptor and string */
2790 1 2 L           FIXED BINARY(15),           /* Length */
2791 1 2 T           FIXED BINARY(7),           /* Type */
2792 1 2 C           FIXED BINARY(7),           /* Class */
2793 1 2 A           POINTER;                     /* Address */
2794 1
2795 1
2796 1 Declare
2797 1 COMM_STR      BASED(COMM_STR_PTR),         /* Comment descriptor and string */
2798 1 2 L           FIXED BINARY(31),           /* Comment string length */
2799 1 2 A           POINTER,                     /* Comment string address */
2800 1 2 S           CHAR(MNR_HDR$K_MAXCOMLEN); /* Comment string */
2801 1
```

MONMAIN
V04-000

H 13
16-SEP-1984 02:11:08
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (27)

Page 35

2802 1
2803 1
2804 1
2805 1
2806 1
2807 1
2808 1
2809 1
2810 1
2811 1

Declare
1 Q_VAL,
2 L
2 A

Q_VAL_TIME
Q_VAL_SECS
TS_LEN

FIXED BINARY(31),
POINTER,

CHAR(TIME TOK SIZE),
CHAR(SECONDS TOK SIZE+1),
FIXED BINARY(15);

/* String descriptor for qualifier value */
/* Length */
/* Address */

/* Qualifier value string for time values */
/* Qualifier value string for seconds values */
/* Actual length of time spec */

```
2812 1 QUAL_SPECIFIED = NO; /* No qualifiers explicitly specified yet */
2813 1 CALL = SYS$GETTIM(MC->MCASQ_CURR_TIME); /* Get current time from system */
2814 1
2815 1 Q_VAL.L = LENGTH(Q_VAL_TIME); /* Set length field of descriptor */
2816 1 Q_VAL.A = ADDR(Q_VAL_TIME); /* Set address field of descriptor */
2817 1
2818 1 CALL = CLIS$PRESENT(QUAL$$_BEG); /* Get BEGINNING qualifier presence indicator */
2819 1 IF CALL = CLIS$_PRESENT /* If explicitly present, */
2820 1 THEN
2821 1 DO;
2822 1 QUAL_SPECIFIED = YES; /* Indicate qualifier explicitly specified */
2823 1 IF CLIS$GET_VALUE(QUAL$$_BEG,Q_VAL,TS_LEN) /* Get 'BEGINNING' string and check status */
2824 1 THEN DO; /* Value was specified */
2825 1 Q_VAL.L = TS_LEN; /* Pick up actual length */
2826 1 IF Q_VAL.L > TIME_TOK_SIZE /* Check for valid size for time spec */
2827 1 THEN DO;
2828 1 CALL MON_ERR(MNRS_INVBTIMSP); /* Log possible error */
2829 1 RETURN(MNRS_INVBTIMSP); /* ... and return with status */
2830 1 END;
2831 1 IF ^ STR$UPCASE(Q_VAL,Q_VAL) /* Upcase and check status */
2832 1 THEN DO;
2833 1 CALL MON_ERR(MNRS_INVBTIMSP); /* Log possible error */
2834 1 RETURN(MNRS_INVBTIMSP); /* ... and return with status */
2835 1 END;
2836 1 IF ^ LIB$CVT_TIME(Q_VAL,M->MRB$Q_BEGINNING) /* Cvt to system time */
2837 1 THEN DO;
2838 1 CALL MON_ERR(MNRS_INVBTIMSP); /* Log possible error */
2839 1 RETURN(MNRS_INVBTIMSP); /* ... and return with status */
2840 1 END;
2841 1 END;
2842 1 ELSE
2843 1 M->MRB$Q_BEGINNING = DEF_MRBPTR->MRB$Q_BEGINNING; /* Value was defaulted */
2844 1 END; /* Store the default value */
2845 1
2846 1 Q_VAL.L = LENGTH(Q_VAL_TIME); /* Set length field of descriptor */
2847 1 CALL = CLIS$PRESENT(QUAL$$_END); /* Get ENDING qualifier presence indicator */
2848 1 IF CALL = CLIS$_PRESENT /* If explicitly present, */
2849 1 THEN
2850 1 DO;
2851 1 QUAL_SPECIFIED = YES; /* Indicate qualifier explicitly specified */
2852 1 IF CLIS$GET_VALUE(QUAL$$_END,Q_VAL,TS_LEN) /* Get 'ENDING' string and check status */
2853 1 THEN DO; /* Value was specified */
2854 1 Q_VAL.L = TS_LEN; /* Pick up actual length */
2855 1 IF Q_VAL.L > TIME_TOK_SIZE /* Check for valid size for time spec */
2856 1 THEN DO;
2857 1 CALL MON_ERR(MNRS_INVETIMSP); /* Log possible error */
2858 1 RETURN(MNRS_INVETIMSP); /* ... and return with status */
2859 1 END;
2860 1 IF ^ STR$UPCASE(Q_VAL,Q_VAL) /* Upcase and check status */
2861 1 THEN DO;
2862 1 CALL MON_ERR(MNRS_INVETIMSP); /* Log possible error */
2863 1 RETURN(MNRS_INVETIMSP); /* ... and return with status */
2864 1 END;
2865 1
```

MONMAIN
V04-000

J 13
16-SEP-1984 02:11:09
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273 Page 37
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (28)

2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879

4
3
4
4
3
2
2
2
2
2
1

```
                END;  
IF ^ LIB$CVT_TIME(Q_VAL,M->MRB$Q_ENDING)  
  THEN DO;  
        CALL MON_ERR(MNRS_INVETIMSP);  
        RETURN(MNRS_INVETIMSP);  
        END;  
  END;  
ELSE  
  M->MRB$Q_ENDING = DEF_MRBPTR->MRB$Q_ENDING;  
END;
```

```
/* Cvt to system time */  
/* Log possible error */  
/* ... and return with status */  
  
/* Value was defaulted */  
/* Store the default value */
```

```
2880 1 Q_VAL.L = LENGTH(Q_VAL_SECS);
2881 1 Q_VAL.A = ADDR(Q_VAL_SECS);
2882 1
2883 1 CALL = CLISP$PRESENT(QUAL$$_INT);
2884 1 IF CALL = CLISP$$_PRESENT
2885 1 THEN
2886 1 DO;
2887 2 QUAL_SPECIFIED = YES;
2888 2 IF CLISP$$_GET_VALUE(QUAL$$_INT,Q_VAL)
2889 2 THEN DO;
2890 3 Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;
2891 3 IF Q_VAL.L <= 0
2892 3 THEN DO;
2893 4 CALL MON_ERR(MNRS_INVINTSP);
2894 4 RETURN(MNRS_INVINTSP);
2895 4 END;
2896 3 CURR_ERRCODE = MNRS_INVINTSP;
2897 3 M->MRB$$_INTERVAL = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31);
2898 3 CURR_ERRCODE = 0;
2899 3 IF M->MRB$$_INTERVAL <= 0
2900 3 THEN DO;
2901 4 CALL MON_ERR(MNRS_INVINTSP);
2902 4 RETURN(MNRS_INVINTSP);
2903 4 END;
2904 3 END;
2905 2 ELSE
2906 2 M->MRB$$_INTERVAL = DEF_MRBPTR->MRB$$_INTERVAL;
2907 2 END;
2908 1
2909 1 Q_VAL.L = LENGTH(Q_VAL_SECS);
2910 1 CALL = CLISP$PRESENT(QUAL$$_FLUSH);
2911 1 IF CALL = CLISP$$_PRESENT
2912 1 THEN
2913 1 DO;
2914 2 QUAL_SPECIFIED = YES;
2915 2 IF CLISP$$_GET_VALUE(QUAL$$_FLUSH,Q_VAL)
2916 2 THEN DO;
2917 3 Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;
2918 3 IF Q_VAL.L <= 0
2919 3 THEN DO;
2920 4 CALL MON_ERR(MNRS_INVFLUSHSP);
2921 4 RETURN(MNRS_INVFLOSHSP);
2922 4 END;
2923 3 CURR_ERRCODE = MNRS_INVFLUSHSP;
2924 3 M->MRB$$_FLUSH = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31);
2925 3 CURR_ERRCODE = 0;
2926 3 IF M->MRB$$_FLUSH <= 0
2927 3 THEN DO;
2928 4 CALL MON_ERR(MNRS_INVFLUSHSP);
2929 4 RETURN(MNRS_INVFLOSHSP);
2930 4 END;
2931 3 END;
2932 2 ELSE
2933 2
```

/* Set length field of descriptor */
/* Set address field of descriptor */
/* Get INTERVAL qualifier presence indicator
/* If explicitly present, */
/* Indicate qualifier explicitly specified */
/* Get "INTERVAL" string and check status */
/* Value was specified */
/* Eliminate trailing blanks */
/* Check for valid size for "seconds" */
/* Log possible error */
/* ... and return with status */
/* Set MONITOR code in case conversion error
/* Convert seconds to binary */
/* Reset to default MONITOR code */
/* Check for valid value */
/* Log possible error */
/* ... and return with status */
/* Value was defaulted */
/* Store the default value */
/* Set length field of descriptor */
/* Get FLUSH qualifier presence indicator */
/* If explicitly present, */
/* Indicate qualifier explicitly specified */
/* Get "FLUSH" string and check status */
/* Value was specified */
/* Eliminate trailing blanks */
/* Check for valid size for "seconds" */
/* Log possible error */
/* ... and return with status */
/* Set MONITOR code in case conversion error
/* Convert seconds to binary */
/* Reset to default MONITOR code */
/* Check for valid value */
/* Log possible error */
/* ... and return with status */
/* Value was defaulted */

```
2936      M->MRB$$_FLUSH = DEF_MRBPTR->MRB$$_FLUSH;
2937      END;
2938
2939      Q_VAL.L = LENGTH(Q_VAL_SECS);
2940
2941      CALL = CLIS$PRESENT(QUAL$$_VIEW);
2942      IF CALL = CLIS$_PRESENT
2943      THEN
2944      DO;
2945          QUAL SPECIFIED = YES;
2946          IF CLIS$GET_VALUE(QUAL$$_VIEW,Q_VAL)
2947          THEN DO;
2948              Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;
2949              IF Q_VAL.L <= 0
2950              THEN DO;
2951                  CALL MON_ERR(MNRS$_INVVIEWSP);
2952                  RETURN(MNRS$_INVVIEWSP);
2953                  END;
2954                  CURR_ERRCODE = MNRS$_INVVIEWSP;
2955                  M->MRB$$_VIEWING_TIME = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31);
2956                  CURR_ERRCODE = 0;
2957                  IF M->MRB$$_VIEWING_TIME <= 0
2958                  THEN DO;
2959                      CALL MON_ERR(MNRS$_INVVIEWSP);
2960                      RETURN(MNRS$_INVVIEWSP);
2961                      END;
2962              END;
2963          ELSE
2964              M->MRB$$_VIEWING_TIME = DEF_MRBPTR->MRB$$_VIEWING_TIME;
2965          END;
2966      END;
2967
2968
```

```
/* Store the default value */

/* Set length field of descriptor */

/* Get VIEWING_TIME qualifier presence indic
/* If explicitly present, */

/* Indicate qualifier explicitly specified *
/* Get "VIEWING_TIME" string and check statu
/* Value was specified */
/* Eliminate trailing blanks */
/* Check for valid size for "seconds" */

/* Log possible error */
/* ... and return with status */

/* Set MONITOR code in case conversion error
/* Convert seconds to binary */
/* Reset to default MONITOR code */
/* Check for valid value */

/* Log possible error */
/* ... and return with status */

/* Value was defaulted */
/* Store the default value */
```

```
2969 1 CALL = CLIS$PRESENT(QUAL$SL_BY_NODE);
2970 1 IF CALL = CLIS$_PRESENT
2971 1 THEN
2972 1 DO;
2973 2 QUAL SPECIFIED = YES;
2974 2 M->MRBSV_BY_NODE = YES;
2975 2 END;
2976 1
2977 1 IF CALL = CLIS$_NEGATED
2978 1 THEN DO;
2979 2 QUAL SPECIFIED = YES;
2980 2 M->MRBSV_BY_NODE = NO;
2981 2 END;
2982 1
2983 1 CALL = CLIS$PRESENT(QUAL$SL_INP);
2984 1 IF CALL = CLIS$_PRESENT
2985 1 THEN DO;
2986 2 QUAL SPECIFIED = YES;
2987 2 M->MRBSV_MFSUM = NO;
2988 2 CALL = BUILD_IFB_TABLE();
2989 2 IF ^STATUS
2990 2 THEN RETURN(CALL);
2991 2 END;
2992 1 IF CALL = CLIS$_NEGATED
2993 1 THEN DO;
2994 2 QUAL SPECIFIED = YES;
2995 2 M->MRBSV_MFSUM = NO;
2996 2 M->MRBSA_INPUT = NULL();
2997 2 END;
2998 1
```

/* Get BY_NODE qualifier presence indicator
/* If explicitly present, */

/* Indicate qualifier explicitly specified *
/* Turn on flag in MRB */

/* If explicitly negated, */

/* Indicate qualifier explicitly specified *
/* Turn off flag in MRB */

/* Get INPUT qualifier presence indicator */
/* If explicitly present, */

/* Indicate qualifier explicitly specified *
/* Set to NO for now (may change in BUILD_IF
/* Build the table of Input File Blocks (IFB

/* return with error status, MON_ERR was alr

/* If explicitly negated, */

/* Indicate qualifier explicitly specified *
/* Turn off Multi-File Summary indicator */
/* Indicate no input file */

```
2999      CALL = CLISPRESNT(QUAL$$_DISP);
3000      IF CALL = CLIS$_PRESENT
3001          THEN DO;
3002          QUAL SPECIFIED = YES;
3003          IF DISP_PTR_VOL = NULL()
3004              THEN DO;
3005              ALLOCATE FILE_SPEC;
3006              DISP_PTR_VOL = FILE_SPEC_PTR;
3007              FILE_SPEC.L = LENGTH(FILE_SPEC.S);
3008              FILE_SPEC.A = ADDR(FILE_SPEC.S);
3009              END;
3010
3011          ELSE DO;
3012              FILE_SPEC_PTR = DISP_PTR_VOL;
3013              FILE_SPEC.L = FILE_SPEC_SIZE;
3014              END;
3015
3016          IF CLISGET VALUE(QUAL$$_DISP,FILE_SPEC)
3017              THEN DO;
3018              DISP_PTR_SWAP = YES;
3019              TEMP = INDEX(FILE_SPEC.S,' ') - 1;
3020              IF TEMP >= 0 THEN FILE_SPEC.L = TEMP;
3021              M->MRB$$_DISPLAY = FILE_SPEC_PTR;
3022              M->MRB$$_DISP_TO_FILE = YES;
3023              END;
3024
3025          ELSE DO;
3026              M->MRB$$_DISPLAY = ADDR(DEF$$_DISP);
3027              M->MRB$$_DISP_TO_FILE = NO;
3028              END;
3029
3030          END;
3031
3032      IF CALL = CLIS$_NEGATED
3033          THEN DO;
3034          QUAL SPECIFIED = YES;
3035          M->MRB$$_DISPLAY = NULL();
3036          M->MRB$$_DISP_TO_FILE = NO;
3037          END;
3038
3039      CALL = CLISPRESNT(QUAL$$_REC);
3040      IF CALL = CLIS$_PRESENT
3041          THEN DO;
3042          QUAL SPECIFIED = YES;
3043          IF REC_PTR_VOL = NULL()
3044              THEN DO;
3045              ALLOCATE FILE_SPEC;
3046              REC_PTR_VOL = FILE_SPEC_PTR;
3047              FILE_SPEC.L = LENGTH(FILE_SPEC.S);
3048              FILE_SPEC.A = ADDR(FILE_SPEC.S);
3049              END;
3050
3051          ELSE FILE_SPEC_PTR = REC_PTR_VOL;
3052
3053          IF CLISGET VALUE(QUAL$$_REC,FILE_SPEC)
3054              THEN DO;
```

```
/* Get DISPLAY qualifier presence indicator
/* If explicitly present, */

/* Indicate qualifier explicitly specified *
/* If no volatile file spec string area, */

/* then get one */
/* set up a ptr to it */
/* set length ... */
/* and address in descriptor */

/* Otherwise, simply point to existing one *
/* ... and re-init its length */

/* Qualifier value specified ? */

/* Yes -- ind to SET_CMD a ptr swap is neces
/* Find trailing blanks in value (string) */
/* If found one, set new length */
/* Store away pointer to value descr */
/* Indicate filespec specified */

/* No -- qualifier value defaulted */
/* Store a default value descr */
/* ... and default indicator */

/* If explicitly negated, */

/* Indicate qualifier explicitly specified *
/* Indicate no display output */
/* ..... */

/* Get RECORD qualifier presence indicator *
/* If explicitly present, */

/* Indicate qualifier explicitly specified *
/* If no volatile file spec string area, */

/* then get one */
/* set up a ptr to it */
/* set length ... */
/* and address in descriptor */

/* Otherwise, simply point to existing one *
/* Qualifier value specified ? */
```

```
3055      REC_PTR_SWAP = YES;  
3056      M->MRBSA_RECORD = FILE_SPEC_PTR;  
3057      END;  
3058  
3059      ELSE M->MRBSA_RECORD = ADDR(DEF$$_REC);  
3060  
3061      END;  
3062  
3063      IF CALL = CLIS$_NEGATED  
3064      THEN DO;  
3065          QUAL_SPECIFIED = YES;  
3066          M->MRBSA_RECORD = NULL();  
3067          END;  
3068  
3069      CALL = CLIS$_PRESENT(QUAL$_SUMM);  
3070      IF CALL = CLIS$_PRESENT  
3071      THEN DO;  
3072          QUAL_SPECIFIED = YES;  
3073          IF SUMM_PTR_VOL = NULL()  
3074          THEN DO;  
3075              ALLOCATE FILE_SPEC;  
3076              SUMM_PTR_VOL = FILE_SPEC_PTR;  
3077              FILE_SPEC.L = LENGTH(FILE_SPEC.S);  
3078              FILE_SPEC.A = ADDR(FILE_SPEC.S);  
3079              END;  
3080  
3081              ELSE FILE_SPEC_PTR = SUMM_PTR_VOL;  
3082  
3083              IF CLIS$_GET_VALUE(QUAL$_SUMM, FILE_SPEC)  
3084              THEN DO;  
3085                  SUMM_PTR_SWAP = YES;  
3086                  M->MRBSA_SUMMARY = FILE_SPEC_PTR;  
3087                  END;  
3088  
3089                  ELSE M->MRBSA_SUMMARY = ADDR(DEF$$_SUMM);  
3090  
3091                  END;  
3092  
3093              IF CALL = CLIS$_NEGATED  
3094              THEN DO;  
3095                  QUAL_SPECIFIED = YES;  
3096                  M->MRBSA_SUMMARY = NULL();  
3097                  END;  
3098  
3099              CALL = CLIS$_PRESENT(QUAL$_COMM);  
3100              IF CALL = CLIS$_PRESENT  
3101              THEN DO;  
3102                  QUAL_SPECIFIED = YES;  
3103                  IF COMM_PTR_VOL = NULL()  
3104                  THEN DO;  
3105                      ALLOCATE COMM_STR;  
3106                      COMM_PTR_VOL = COMM_STR_PTR;  
3107                      END;  
3108  
3109                      ELSE COMM_STR_PTR = COMM_PTR_VOL;  
3110  
3111                      IF CLIS$_GET_VALUE(QUAL$_COMM, DYN_STRING)
```

```
/* Yes -- ind to SET_CMD a ptr swap is neces  
/* Store away pointer to string descr */
```

```
/* No -- store a default value */
```

```
/* If explicitly negated, */
```

```
/* Indicate qualifier explicitly specified *  
/* Indicate no record output */
```

```
/* Get SUMMARY qualifier presence indicator  
/* If explicitly present, */
```

```
/* Indicate qualifier explicitly specified *  
/* If no volatile file spec string area, */
```

```
/* then get one */  
/* set up a ptr to it */  
/* set length ... */  
/* and address in descriptor */
```

```
/* Otherwise, simply point to existing one *
```

```
/* Qualifier value specified ? */
```

```
/* Yes -- ind to SET_CMD a ptr swap is neces  
/* Store away pointer to string descr */
```

```
/* No -- store a default value */
```

```
/* If explicitly negated, */
```

```
/* Indicate qualifier explicitly specified *  
/* Indicate no summary output */
```

```
/* Get COMMENT qualifier presence indicator  
/* If explicitly present, */
```

```
/* Indicate qualifier explicitly specified *  
/* If no volatile comment string area, */
```

```
/* then get one ... */  
/* and set up a ptr to it */
```

```
/* Otherwise, simply point to existing one *
```

```
/* Qualifier value specified ? */
```

```
3112      THEN DO;
3113          COMM_STR.S = DYN_STRING_S;
3114          COMM_STR.L = DYN_STRING.L;
3115          COMM_STR.A = ADDR(COMM_STR.S);
3116          COMM_PTR_SWAP = YES;
3117          M->MRB$A_COMMENT = COMM_STR_PTR;
3118          END;
3119      ELSE M->MRB$A_COMMENT = NULL();
3120
3121      END;
3122
3123  IF CALL = CLIS_NEGATED
3124      THEN DO;
3125      QUAL_SPECIFIED = YES;
3126      M->MRB$A_COMMENT = NULL();
3127      END;
3128
3129  RETURN(NORMAL);
3130
3131
```

/* Yes -- move string out of dyn area */
/* ... and set up its length */
/* ... and address */
/* Ind to SET_CMD a ptr swap is necessary */
/* Store away pointer to string descr */

/* No -- store a default value */

/* If explicitly negated, */

/* Indicate qualifier explicitly specified */
/* Indicate no comment string */

/* Return with status */

```
3132 1 BUILD_IFB_TABLE: Procedure Returns(Fixed Binary(31));
3133
3134 2 /*++
3135 3 /*
3136 4 /* FUNCTIONAL DESCRIPTION:
3137 5 /*
3138 6 /* BUILD_IFB_TABLE
3139 7 /*
3140 8 /* This routine builds the IFB (Input File Block) TABLE.
3141 9 /* In addition, it sets up MRB$A_INPUT to point to the IFB TABLE, and
3142 10 /* sets up MRB$B_INP_FILES to be the number of input files described by
3143 11 /* the table.
3144 12 /*
3145 13 /* INPUTS:
3146 14 /*
3147 15 /* None
3148 16 /*
3149 17 /* IMPLICIT INPUTS:
3150 18 /*
3151 19 /* IFB_TABLE, MRB
3152 20 /*
3153 21 /* OUTPUTS:
3154 22 /*
3155 23 /* None
3156 24 /*
3157 25 /* IMPLICIT OUTPUTS:
3158 26 /*
3159 27 /* IFB_TABLE built, MRB$A_INPUT and MRB$B_INP_FILES established.
3160 28 /*
3161 29 /* ROUTINE VALUE:
3162 30 /*
3163 31 /* Normal, or bad status from LIB$FIND_FILE
3164 32 /*
3165 33 /* SIDE EFFECTS:
3166 34 /*
3167 35 /* None
3168 36 /*
3169 37 /*/
3170 2
```


MONMAIN
V04-000

F 14
16-SEP-1984 02:11:14
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (33)

Page 46

```
3994      2      1      FIXED BINARY(7),  
3995      2      VALUE FOUND      BIT(1) ALIGNED,  
3996      2      IFBPTR      POINTER,  
3997      2      IFB_TAB_PTR      POINTER;  
3998      2  
3999      2      Declare  
4000      2      1 IFB_TABLE      BASED(IFB_TAB_PTR),  
4001      2      2 AN_IFB      (1:MAX_INP_FILES+1) CHAR(IFB$K_SIZE);  
4002      2  
4003      2      Declare  
4004      2      REC_DEF_S      CHAR(DEF$L_REC) BASED(DEF$A_REC);  
4005      2  
/* Loop control */  
/* YES => a value for the /INPUT qualifier found */  
/* Pointer to Input File Block (IFB) */  
/* Pointer to IFB_TABLE */  
/* Input File Block (IFB) Table */  
/* A single IFB */  
/* String for default recording file spec */
```

```
4006      IF IFB_TAB_VOL = NULL()
4007      THEN DO;
4008          ALLOCATE IFB_TABLE;
4009          IFB_TAB_VOL = IFB_TAB_PTR;
4010          DO I = 1 TO MAX_INP_FILES + 1;
4011              UNSPEC(AN_IFB(I)) = '0'B;
4012          END;
4013      ELSE IFB_TAB_PTR = IFB_TAB_VOL;
4014
4015      /*
4016      NOTE -- at this point, the volatile pointer (IFB_TAB_VOL) and the base
4017      pointer (IFB_TAB_PTR) both point to the IFB_TABLE in use.
4018      */
4019      M->MRBSB_INP_FILES = 0;
4020      VALUE_FOUND = YES;
4021      CALL = $$$_NORMAL;
4022      I = 1;
4023      CONTEXT = 0;
4024
4025      /*
4026      NOTE -- What follows are two loops - the outer loop does CLISGET_VALUE calls, and the inner loop does
4027      LIB$FIND_FILE calls. The CLISGET_VALUE loop is controlled by STATUS, which is the low bit of CALL, a
4028      also by VALUE_FOUND. CALL will always be $$$_NORMAL unless the LIB$FIND_FILE loop runs into trouble.
4029      CALL is what is ultimately returned by this procedure.
4030      */
4031      DO WHILE(STATUS & VALUE_FOUND);
4032          FIND_FILE_CALL = $$$_NORMAL;
4033          DYN_SPEC.T = DSC$K_DTYPE_T;
4034          DYN_SPEC.C = DSC$K_CLASS_D;
4035          IF *CLISGET_VALUE(QUAL$INP,DYN_SPEC)
4036          THEN DO;
4037              IF M->MRBSB_INP_FILES = 0
4038              THEN CALL LIB$COPY_DXDX(DEF$REC,DYN_SPEC);
4039              ELSE VALUE_FOUND = NO;
4040          END;
4041          DO WHILE(STATUS & FIND_FILE_STAT & VALUE_FOUND);
4042              IFB_PTR = ADDR(AN_IFB(I));
4043              IF IFB$A_INPUT = NULL()
4044              THEN DO;
4045                  ALLOCATE PARSED_SPEC;
4046                  IFB$A_INPUT = PARSED_SPEC_PTR;
4047                  PARSED_SPEC.T = DSC$K_DTYPE_T;
4048                  PARSED_SPEC.C = DSC$K_CLASS_D;
4049                  PARSED_SPEC.A = NULL();
4050                  PARSED_SPEC.L = 0;
4051              END;
4052              ELSE PARSED_SPEC_PTR = IFB$A_INPUT;
4053              FIND_FILE_CALL = LIB$FIND_FILE(DYN_SPEC,PARSED_SPEC,CONTEXT,DEF$REC,,,USER_FLAGS);
4054              IF FIND_FILE_STAT
4055              THEN DO;
4056                  M->MRBSB_INP_FILES = M->MRBSB_INP_FILES + 1;
4057                  IF I > MAX_INP_FILES
4058                  THEN DO;
4059
4060
```

4061	6	CALL MON ERR(MNRS TOOMNYFILES);	/* then log the error */
4062	6	CALL = MNRS_TOOMNYFILES;	/* set bad status to end both inner and outer loops
4063	6	END;	
4064	5	ELSE I = I + 1;	/*...else increment file count and continue the loop
4065	5	END;	
4066	4	END;	/* end of LIB\$FIND_FILE loop */
4067	3	IF (^FIND_FILE_STAT & FIND_FILE_CALL ^= RMSS_NMF)	/* "no-more-files" is the only valid error from LIB\$
4068	3	THEN DO;	/* ...any other status means trouble (such as file n
4069	4	CALL MON ERR(MNRS OPENIN,FIND_FILE_CALL,PARSED_SPEC);	/* log error */
4070	4	CALL = MNRS_OPENIN;	/* set bad status to end both the inner and outer lo
4071	4	END;	
4072	3	END;	/* End of CLISGET_VALUE loop */
4073	2	CALL LIB\$FIND_FILE_END(CONTEXT);	/* wipe out context of prev LIB\$FIND_FILE calls */
4074	2	IFB TAB SWAP = YES;	/* Ind to SET_CMD a ptr swap is necessary */
4075	2	M->MRBSA_INPUT = IFB TAB_PTR;	/* Store ptr to IFB table */
4076	2	IF M->MRBSB_INP_FILES > 1 THEN M->MRBSV_MFSUM = YES;	/* If more than 1 file spec, indicate multi-file sum
4077	2		
4078	2	RETURN (CALL);	/* Return to caller */
4079	2		
4080	1	END BUILD_IFB_TABLE;	
4081	1		
4082	1	END GET_QUALIFIERS;	
4083			

```
4084 GET_CLASSES: Procedure (AT_LEAST_ONE_CLASS) Returns(Fixed Binary(31));
4085
4086 /*++
4087 /*
4088 /* FUNCTIONAL DESCRIPTION:
4089 /*
4090 /*     GET_CLASSES
4091 /*
4092 /*     Communicate with CLE to get class_name keywords; then use keywords as
4093 /*     input to LOOKUP_KEY and get back class numbers. Ultimate output is
4094 /*     MRBSO_CLASSBITS, a bit string for which each bit number corresponds
4095 /*     to class number. If a lookup error is found, bad status is returned
4096 /*     to the issuer.
4097 /*
4098 /* INPUTS:
4099 /*
4100 /*     TBS
4101 /*
4102 /* IMPLICIT INPUTS:
4103 /*
4104 /*     TBS
4105 /*
4106 /* OUTPUTS:
4107 /*
4108 /*     TBS
4109 /*
4110 /* IMPLICIT OUTPUTS:
4111 /*
4112 /*     TBS
4113 /*
4114 /* ROUTINE VALUE:
4115 /*
4116 /*     TBS
4117 /*
4118 /* SIDE EFFECTS:
4119 /*
4120 /*     TBS
4121 /*
4122 /*/
4123
```

```
4124 1 /*
4125 1 /*
4126 1 /*
4127 1 /*
4128 1 /*
4129 1 /*
4130 1 /*/
4131 1
4132 1 Declare
4133 1 AT_LEAST_ONE_CLASS BIT(1) ALIGNED, /* YES => user requested at least one class */
4134 1 CALL FIXED BINARY(31), /* Holds function value (return status) of called ro
4135 1 STATUS BIT(1) BASED(ADDR(CALL)), /* Low-order status bit for called routines */
4136 1 CLASS_KEY FIXED BINARY(31) STATIC, /* Class keyword number */
4137 1 /* Note -- must be STATIC to get error msg */
4138 1 TEMP FIXED BINARY(31); /* Scratch "register" */
4139 1
4140 1
4141 1 %REPLACE NOT_SUCCESSFUL BY '0'B; /* Failing severity code */
4142 1 %REPLACE CLASS_TOK_SIZE BY 20; /* Size of token for class-name */
4143 1 %REPLACE OR_OP BY '0111'B; /* OR Boolean operation */
4144 1
4145 1 /*
4146 1 /*
4147 1 /*
4148 1 /*
4149 1 /*
4150 1 /*
4151 1 /*/
4152 1
4153 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
4154 1
4155 1 Declare
4156 1 CLISGET_VALUE ENTRY(ANY, ANY) /* CLE routine to get qualifier values */
4157 1 RETURNS(BIT(1)),
4158 1 LIB$LOOKUP_KEY EXTERNAL ENTRY(ANY, ANY, FIXED BINARY(31)) /* RTL rtn to look up class-names in a table */
4159 1 RETURNS(BIT(1)),
4160 1 MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE); /* MONITOR MACRO-32 routine to log synchronous error
4161 1
4162 1 Declare
4163 1 I FIXED BINARY(7), /* Loop control */
4164 1 MAX_CLASS_NO FIXED BINARY(31) GLOBALREF VALUE, /* Maximum defined class number */
4165 1 ALL_CLSNO FIXED BINARY(31) GLOBALREF VALUE, /* ALL class Pseudo class number */
4166 1 ALL_CLASS_FOUND BIT(1), /* Flag to indicate ALL class found on command line
4167 1 MNR$ INVCSNM FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
4168 1 NORMAL FIXED BINARY(31) GLOBALREF, /* MONITOR normal return status */
4169 1 CDBPTR POINTER GLOBALREF, /* Pointer to CDB (Class Descriptor Block) */
4170 1 C POINTER DEFINED(CDBPTR), /* Short-hand synonym for CDBPTR */
4171 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
4172 1 M POINTER DEFINED(MRBPTR), /* Short-hand synonym for MRBPTR */
4173 1 QUALPTR POINTER GLOBALREF; /* Pointer to Qualifier Descriptors Block */
4174 1
4175 1 Declare
4176 1 1 CDBHEAD GLOBALREF, /* Table of CDB's */
4177 1 2 CDBLOCK (0:127) CHAR(CDB$K_SIZE);
4178 1
4179 1 Declare
```

```
4947 1      CLASSTABLE      CHAR(1)      GLOBALREF;      /* Table of class names & numbers (dummy) */
4948 :      :      :      :      /* Note -- CLASSTABLE is declared here simply */
4949 :      :      :      :      /* so its address can be referenced below */
4950
4951 Declare
4952 REQUEST_CLASS_MASK      BIT(MAX_CLASS_NO+1),      /* Requested classes have bits on */
4953 REQUEST_CLASS_VEC      (0:MAX_CLASS_NO) BIT(1) DEFINED(REQUEST_CLASS_MASK);
4954 :      :      :      :      /* Alias for REQUEST_CLASS_MASK, but bit-addressable */
4955
4956 Declare
4957 01 CLASS_VAL STATIC,      /* String descr for value of CLASS NAME qualifier */
4958 :      :      :      :      /* Note -- must be STATIC to get INVCLSNM msg */
4959      02 L      FIXED BINARY(31),      /* Length */
4960      02 A      POINTER,      /* Address */
4961      02 S      CHAR(CLASS_TOK_SIZE+1);      /* String */
4962
4963 CLASS_VAL.L = LENGTH(CLASS_VAL.S);      /* Init length longword of descr */
4964 CLASS_VAL.A = ADDR(CLASS_VAL.S);      /* Init address longword of descr */
4965
4966 REQUEST_CLASS_MASK = '0'B;      /* Turn off all class bits initially */
4967 ALL_CLASS_FOUND = NO;      /* Assume we won't find ALL pseudo-class on this command lin
4968 DO WHILE(CLISGET_VALUE(QUAL$CLASS,CLASS_VAL));      /* Loop once for each requested class */
4969     CLASS_VAL.L = INDEX(CLASS_VAL.S,' ') = 1;      /* Now strip off trailing blanks */
4970     IF CLASS_VAL.L < 0 THEN CLASS_VAL.L = CLASS_TOK_SIZE; /* If too long, replace with max token size */
4971
4972     IF ^ LIB$LOOKUP_KEY(CLASS_VAL,CLASSTABLE,CLASS_KEY) /* Get class keyword number */
4973     THEN DO;
4974         CALL MON ERR(MNR$ INVCLSNM,,CLASS_VAL);      /* Log error if bad class name */
4975         RETURN(MNR$ INVCLSNM);      /* ... and return with status */
4976     END;
4977     CLASS_VAL.L = CLASS_TOK_SIZE + 1;      /* Restore string len for next loop */
4978     IF CLASS_KEY = ALL_CLSN0      /* If all classes */
4979     THEN DO;
4980         ALL_CLASS_FOUND = YES;      /* indicate we found ALL on this command line */
4981         M->MRBSV_ALL_CLASS = YES;      /* indicate its ALL class*/
4982         DO I = 0 TO MAX_CLASS_NO;      /* Loop once for each possible class */
4983             REQUEST_CLASS_VEC(I) = YES;      /* Turn on bit for this class */
4984             CDBPTR = ADDR(CDBLOCK(I));      /* Get CDB addressability */
4985             IF C->CDBSV_DISABLE THEN REQUEST_CLASS_VEC(I) = NO; /* If this class disabled, then ignore it */
4986         END;
4987     ELSE REQUEST_CLASS_VEC(CLASS_KEY) = YES;      /* Turn on bit for this class */
4988     CDBPTR = ADDR(CDBLOCK(CLASS_KEY));      /* Get CDB addressability */
4989     IF C->CDBSV_DISABLE      /* If this class disabled, */
4990     THEN REQUEST_CLASS_VEC(CLASS_KEY) = NO;      /* then ignore it */
4991     ELSE DO;      /* Otherwise, */
4992         CALL = GET_CLASS_QUALS(CLASS_KEY);      /* Process class qualifiers for this class */
4993         IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* ... and check status */
4994     END;
4995
4996 END;
4997
4998 IF INDEX(REQUEST_CLASS_MASK,'1'B) ^= 0      /* If any classes specified, */
4999 THEN DO;      /* indicate got at least one */
5000     AT_LEAST_ONE_CLASS = YES;      /* if we didn't find the ALL pseudo-class */
5001     IF ALL_CLASS_FOUND = NO      /* ...make sure the ALL class flag is clear (this logic is n
5002     THEN M->MRBSV_ALL_CLASS = NO;
```

MONMAIN
V04-000

L 14
16-SEP-1984 02:11:17 VAX-11 PL/I X2.1-273 Page 52
5-SEP-1984 15:09:57 ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (36)

5004 : 2
5005 2
5006 2
5007 1
5008 2
5009 2
5010 2
5011 2
5012 1
5013 1
5014 1
5015 1

```
      M->MRB$O_CLASSBITS = REQUEST_CLASS_MASK;
      END;
ELSE DO;
      IF INDEX(M->MRB$O_CLASSBITS,'1'B) = 0
      THEN AT_LEAST_ONE_CLASS = NO;
      ELSE AT_LEAST_ONE_CLASS = YES;
      END;
RETURN(NORMAL);
```

/* for defaulting to work correctly in interactive mode) */
/* ... move selected bits */
/* No classes specified, use MRB\$O_CLASSBITS */
/* If no classes specified there, */
/* then indicate so */
/* else indicate got one or more */
/* Return to caller */

```
5016 1 GET_CLASS_QUALS: Procedure (CLSNO) Returns (fixed binary(31));
5017
5018 Declare
5019 CLISPRESENT ENTRY(ANY) RETURNS(BIT(1)), /* CLE routine to determine presence of quals */
5020 CLISGET_VALUE ENTRY(ANY, ANY, FIXED BINARY(15)) /* CLE routine to get qualifier values */
5021 RETURNS(BIT(1));
5022
5023 Declare
5024 MNRS_QUALINV FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
5025 MNRS_SQUALERR FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
5026 MNRS_PDQUALERR FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
5027 MNRS_INVITEMNM FIXED BINARY(31) GLOBALREF VALUE; /* Error message code */
5028
5029 Declare
5030 MODES_CLSNO FIXED BINARY(31) GLOBALREF VALUE; /* MODES class number */
5031
5032 Declare
5033 TEMP_CDBPTR POINTER ; /* tmp pointer for ALL class qualifier loop */
5034
5035 Declare
5036 1 CLASSTABLE GLOBALREF, /* Table of class names & numbers */
5037 2 VECTOR_CT FIXED BINARY(31), /* Count of longwords following in table */
5038 2 CL_DESCR (0:127),
5039 3 CL_PTR POINTER, /* Pointer to class cstring */
5040 3 CL_NO FIXED BINARY(31); /* Class number */
5041
5042 Declare
5043 1 STAT_TABLE GLOBALREF, /* Table of pointers to str descs for statistic qua
5044 2 STAT_DESC (0:STATS-1) POINTER;
5045
5046 Declare
5047 1 PROCD_TABLE GLOBALREF, /* Table of ptrs to str descs for PROCESSES display
5048 2 PROCD_DESC (0:PROCDISPS-1) POINTER;
5049
5050 Declare
5051 I FIXED BINARY(7), /* Loop control */
5052 J FIXED BINARY(7), /* Loop control */
5053 QUAL_FOUND BIT(1) ALIGNED, /* NO => haven't seen a qualifier yet */
5054 CLSNO FIXED BINARY(31), /* Class number */
5055 QD CHAR(8) BASED, /* Dummy qualifier string descr */
5056 CLSTR CHAR(1) BASED; /* Dummy first char of class cstring */
5057
5058 %REPLACE ITEM_TOK_SIZE BY 25; /* Size of token for item name */
5059
5060 Declare
5061 REQUEST_ITEM_MASK BIT(16), /* Requested items have bits on */
5062 REQUEST_ITEM_VEC (0:15) BIT(1) DEFINED(REQUEST_ITEM_MASK); /* Alias for REQUEST_ITEM_MASK, but bit-addressable
5063 ;
5064
5065 Declare
5066 01 ITEM_VAL STATIC, /* String descr for value of /ITEM qualifier */
5067 ; /* Note -- must be STATIC to get INVITEMNM msg */
5068 02 L FIXED BINARY(31), /* Length */
5069 02 A POINTER, /* Address */
5070 02 S CHAR(ITEM_TOK_SIZE); /* String */
5071
```

MONMAIN
V04-000

N 14
16-SEP-1984 02:11:18
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (37)

Page 54

5072
5073
5074
5075
5076 ;
5077
5078

2
2
2
2
2
2
2

Declare
IVAL_LEN
ITEM_LTAB
ITEM_KEY

FIXED BINARY(15),
FIXED BINARY(31) BASED,
FIXED BINARY(31);

/* Actual length of item value string */
/* Dummy first longword of item lookup table */
/* Item keyword number */
/* Note -- must be STATIC to get error msg */

```
5079      2
5080      2
5081      2
5082      2
5083      2
5084      2
5085      2
5086      2
5087      2
5088      2
5089      2
5090      2
5091      2
5092      2
5093      2
5094      3
5095      3
5096      3
5097      3
5098      4
5099      4
5100      4
5101      3
5102      3
5103      4
5104      4
5105      4
5106      5
5107      5
5108      6
5109      6
5110      6
5111      6
5112      5
5113      4
5114      4
5115      4
5116      3
5117      4
5118      4
5119      4
5120      3
5121      2

/*
/*      NOTE -- CDBPTR (and its synonym C) has been set up by caller.
*/

/*
/*      Check for the presence of each of the statistics qualifiers
/*      (ALL, CUR, AVE, MIN, MAX). If specified for the non-standard
/*      class (PROCESSES), or if more than one specified, log error
/*      and return.
*/

QUAL_FOUND = NO;                                /* Indicate no statistics qualifiers found yet */

DO I = 0 TO STATS-1;                             /* Loop for each stat qual for this class */

IF CLIS$PRESENT(STAT_DESC(I)->QD)                /* If this stat qual is present, */
  THEN IF C->CDB$V STD = NO                      /* Check if non-standard class */
    THEN DO:                                     /* Non-STD -- stat quals not allowed */
      CALL MON ERR(MNRS_QUALINV,,STAT_DESC(I)->QD,CL_PTR(CLSNO)->CLSTR); /* Log an error */
      RETURN(MNRS_QUALINV);                      /* ... and return with status */
    END;
  ELSE IF QUAL_FOUND = NO                        /* STD class -- If we haven't seen a qual yet, */
    THEN DO:                                     /* Indicate we found one this time */
      QUAL_FOUND = YES;                          /* If ALL classes */
      IF CLSNO = ALL_CLSNO                       /* Loop once for each possible class */
        THEN DO J = 1 TO MAX_CLASS_NO;          /* If this is a standard class */
          IF C->CDB$V STD = YES
            THEN DO:
              TEMP_CDBPTR = ADDR(CDBLOCK(J));    /* ... Get CDB addressability */
              TEMP_CDBPTR->CDB$B_ST = I;          /* ... and move in the requested sta */
              TEMP_CDBPTR->CDB$V_EXPLC = YES;      /* ... also indicate explicit qualif */
            END;
          END;                                  /* End ALL classes logic */
          C->CDB$B_ST = I;                      /* Move in the requested stat code */
          C->CDB$V_EXPLC = YES;                 /* Indicate a class qualifier explicitly specified */
        END;
      ELSE DO:                                  /* We've seen a stat qual for this class already */
        CALL MON ERR(MNRS_SQUALERR,,CL_PTR(CLSNO)->CLSTR); /* Log an error */
        RETURN(MNRS_SQUALERR);                 /* Return with status */
      END;
    END;
  END;

END;
```

```
5122 1 2 /*
5123 1 2 /* Check for the presence of each of the PROCESSES display qualifiers
5124 1 2 /* (TOPCPU, TOPFAULT, TOPDIO, TOPBIO). If specified for a standard
5125 1 2 /* class, or if more than one specified, log error and return.
5126 1 2 /*
5127 1 2
5128 1 2 QUAL_FOUND = NO; /* Indicate no PROCESSES qualifiers found yet */
5129 1 2
5130 1 2 DO I = 0 TO PROCDISPS-1; /* Loop for each PROCESSES qualifier */
5131 1 2
5132 1 2 IF PROCD_DESC(I) ^= NULL() & /* If this proc qual is defined ... */
5133 1 2 CLISPRESNT(PROCD_DESC(I)->QD) /* ... AND it is present, */
5134 1 2 THEN IF C->CDB$V-STD = YES /* Check if standard class */
5135 1 2 THEN DO; /* STD class -- proc quals not allowed */
5136 1 2 CALL MON_ERR(MNR$_QUALINV,,PROCD_DESC(I)->QD,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5137 1 2 RETURN(MNR$_QUALINV); /* ... and return with status */
5138 1 2 END;
5139 1 2 ELSE IF QUAL_FOUND = NO /* Non-STD class -- If we haven't seen a qual yet, */
5140 1 2 THEN DO;
5141 1 2 QUAL_FOUND = YES; /* Indicate we found one this time */
5142 1 2 C->CDB$B-ST = I; /* ... and move in the requested display type */
5143 1 2 END;
5144 1 2 ELSE DO; /* We've seen a proc qual for this class already */
5145 1 2 CALL MON_ERR(MNR$_PDQUALERR,,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5146 1 2 RETURN(MNR$_PDQUALERR); /* Return with status */
5147 1 2 END;
5148 1 2 END;
5149 1 2
```

```
5150 1      /*  
5151 1      /* Check for the presence of the /ITEM class qualifier. If present,  
5152 1      /* call CLISGET_VALUE in a loop to obtain the requested item keywords.  
5153 1      /* Then call LIB$LOOKUP_KEY to get numerical bit values for the keywords  
5154 1      /* and set the bits (in CDX$W_IBITS) corresponding to the bit values.  
5155 1      /*/  
5156 2  
5157 2      IF CLISPRESENT(QUAL$L_ITEM)                                /* If /ITEM qual is present, */  
5158 2      THEN IF C->CDB$V_ROMOG = NO                                /* Check if homogeneous class */  
5159 2      THEN DO;                                                    /* Not homog -- /ITEM qual not allowed */  
5160 2          CALL MON_ERR(MNRS_QUALINV,,QUAL$L_ITEM,CL_PTR(CLSNO)->CLSTR); /* Log an error */  
5161 2          RETURN(MNRS_QUALINV);                                       /* ... and return with status */  
5162 2          END;  
5163 2  
5164 2      ELSE DO;                                                    /* Homog class -- loop picking up items */  
5165 2          ITEM_VAL.L = LENGTH(ITEM_VAL.S);                          /* Init length longword of descr */  
5166 2          ITEM_VAL.A = ADDR(ITEM_VAL.S);                            /* Init address longword of descr */  
5167 2          REQUEST_ITEM_MASK = '0'B;                                /* Turn off all item bits initially */  
5168 2  
5169 2      DO WHILE(CLISGET_VALUE(QUAL$L_ITEM,ITEM_VAL,IVAL_LEN)); /* Loop once for each requested item */  
5170 2          ITEM_VAL.L = IVAL_LEN;                                     /* Get actual string length */  
5171 2  
5172 2          IF ^ LIB$LOOKUP_KEY(ITEM_VAL,C->CDB$A_CDX->CDX$A_ILOOKTAB->ITEM_LTAB,ITEM_KEY)  
5173 2          THEN DO;                                                /* Get item keyword number */  
5174 2              CALL MON_ERR(MNRS_INVITEMNM,,ITEM_VAL); /* Log error if bad item name */  
5175 2              RETURN(MNRS_INVITEMNM);                    /* ... and return with status */  
5176 2              END;  
5177 2  
5178 2          ELSE DO;                                                /* Found value for item keyword */  
5179 2              REQUEST_ITEM_VEC(ITEM_KEY) = YES; /* Turn on bit for this item */  
5180 2              ITEM_VAL.L = LENGTH(ITEM_VAL.S); /* Restore string len for next loop */  
5181 2          END;  
5182 2  
5183 2      END;                                                        /* End of DO WHILE loop */  
5184 2  
5185 2      IF INDEX(REQUEST_ITEM_MASK,'1'B) ^= 0                        /* If any items specified, */  
5186 2      THEN C->CDB$A_CDX->CDX$W_IBITS = REQUEST_ITEM_MASK; /* ... move them into the CDX */  
5187 2  
5188 2      END;  
5189 2
```

```
5190 | 2      /*
5191 |      /* The following Begin-End group sets the CDB$V_PERCENT and CDB$V_CPU bits to
5192 |      /* the proper states, depending on the specifications of the /[NO]PERCENT and
5193 |      /* /[NO]CPU qualifiers.
5194 |      */
5195 |
5196 | BEGIN;
5197 | Declare
5198 | CLIS$PRESENT      EXTERNAL ENTRY(ANY) RETURNS(FIXED BINARY(31)), /* CLE routine to determine presence of qualifiers */
5199 | CLIS$_LOCNEG      FIXED BINARY(31)      GLOBALREF VALUE; /* CLIS$PRESENT return status code for 'explicitly ne
5200 |
5201 |      /*
5202 |      /* Check for presence of /[NO]PERCENT qualifier
5203 |      */
5204 |
5205 | CALL = CLIS$PRESENT(QUAL$$_P$CENT); /* Get 'present' indicators for /PERCENT */
5206 | IF STATUS /* If present, */
5207 | THEN IF C->CDB$V_UNIFORM /* If uniform (/PERCENT allowed for unif only) */
5208 | THEN C->CDB$V_PERCENT = YES; /* ... then indicate percent stats */
5209 | ELSE DO;
5210 | CALL MON_ERR(MNRS$ QUALINV,,QUAL$$_P$CENT,CL_PTR(CLSNO)->CLSTR); /* ... otherwise, log an error */
5211 | RETURN(MNRS$ QUALINV); /* ... and return with status */
5212 | END;
5213 | ELSE IF CALL = CLIS$_LOCNEG THEN C->CDB$V_PERCENT = NO; /* IF /PERCENT explicitly negated, turn it off */
5214 |
5215 |      /*
5216 |      /* Now check for presence of /[NO]CPU qualifier
5217 |      */
5218 |
5219 | CALL = CLIS$PRESENT(QUAL$$_CPU); /* Get 'present' indicators for /CPU */
5220 | IF STATUS /* If present, */
5221 | THEN IF CLSNO = MODES CLSNO /* If MODES class, */
5222 | THEN C->CDB$V_CPU = YES; /* ... indicate CPU-specific display */
5223 | ELSE DO; /* Otherwise, /CPU is not allowed */
5224 | CALL MON_ERR(MNRS$ QUALINV,,QUAL$$_CPU,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5225 | RETURN(MNRS$ QUALINV); /* ... and return with status */
5226 | END;
5227 | ELSE IF CALL = CLIS$_LOCNEG THEN C->CDB$V_CPU = NO; /* IF /CPU explicitly negated, turn it off */
5228 |
5229 | END; /* Terminate Begin-End group */
5230 |
5231 | RETURN(NORMAL);
5232 |
5233 | END GET_CLASS_QUALS;
5234 |
5235 | END GET_CLASSES;
5236 | 1
```

```

5237 MONITOR_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute a MONITOR subcommand */
5238
5239 /*+++
5240 /*
5241 /* FUNCTIONAL DESCRIPTION:
5242 /*
5243 /*     MONITOR_CMD
5244 /*
5245 /*     TBS
5246 /*
5247 /* INPUTS:
5248 /*
5249 /*     TBS
5250 /*
5251 /* IMPLICIT INPUTS:
5252 /*
5253 /*     TBS
5254 /*
5255 /* OUTPUTS:
5256 /*
5257 /*     TBS
5258 /*
5259 /* IMPLICIT OUTPUTS:
5260 /*
5261 /*     TBS
5262 /*
5263 /* ROUTINE VALUE:
5264 /*
5265 /*     TBS
5266 /*
5267 /* SIDE EFFECTS:
5268 /*
5269 /*     TBS
5270 /*
5271 /*/
5272

```

```
5273 1 /*
5274 1 /*
5275 1 /*
5276 1 /*
5277 1 /*
5278 1 /*
5279 1 /*/
5280 1
5281 1 Declare
5282 1 CALL FIXED BINARY(31), /* Holds function value (return status) of called ro
5283 1 STATUS BIT(1) BASED(ADDR(CALL)), /* Low-order status bit for called routines */
5284 1 AT_LEAST_ONE_CLASS BIT(1) ALIGNED, /* YES => user requested at least one class */
5285 1 IFBPTR POINTER; /* Pointer to Input File Block (IFB) */
5286 1
5287 1
5288 1 %REPLACE NOT_SUCCESSFUL BY '0'B; /* Failing severity code */
5289 1
5290 1 /*
5291 1 /*
5292 1 /*
5293 1 /*
5294 1 /*
5295 1 /*
5296 1 /*/
5297 1
5298 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
6066 1
6067 1 Declare
6068 1 MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE), /* MONITOR MACRO-32 routine to log synchronous error
6069 1 MOVE_CLASS_QUALS ENTRY (FIXED BINARY(7)) /* MACRO-32 routine to move class ... */
6070 1 RETURNS(FIXED BINARY(31)), /* ... qualifier values for all classes */
6071 1 GET_CLASSES ENTRY (BIT(1) ALIGNED) RETURNS(FIXED BINARY(31)), /* Routine to get info on all classes for this req
6072 1 GET_QUALIFIERS ENTRY RETURNS(FIXED BINARY(31)), /* Routine to store qualifier info for this request
6073 1 EXECUTE_REQUEST ENTRY RETURNS(FIXED BINARY(31)), /* Routine to execute a single MONITOR request */
6074 1 MFSUM_REQUEST ENTRY RETURNS(FIXED BINARY(31)); /* Routine to execute a single MONITOR request with
6075 1
6076 1 Declare
6077 1 MNRS_NOCLASSES FIXED BINARY(31) GLOBALREF VALUE, /* Message code */
6078 1 CUR_TO_ACT FIXED BINARY(31) GLOBALREF VALUE, /* Code value for MOVE_CLASS_QUALS rtn */
6079 1 ALL_TO_ACT FIXED BINARY(31) GLOBALREF VALUE, /* Code value for MOVE_CLASS_QUALS rtn */
6080 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
6081 1 M POINTER DEFINED(MRBPTR), /* Short-hand synonym for MRBPTR */
6082 1 CURR MRBPTR POINTER GLOBALREF, /* Pointer to "current" MRB (Monitor Request Block)
6083 1 ACT MRBPTR POINTER GLOBALREF, /* Pointer to "active" MRB (Monitor Request Block) */
6084 1 FIRST_MON_CMD BIT(1) ALIGNED GLOBALREF, /* YES => first MONITOR (DCL-level) cmd executing */
6085 1 QUAL_SPECIFIED BIT(1) ALIGNED GLOBALREF, /* YES => at least 1 qualifier explicitly specified
6086 1 PROMPT BIT(1) ALIGNED GLOBALREF; /* YES => prompt user for another subcommand */
6087 1
```

```
6088 1 CALL = MOVE_CLASS_QUALS(CUR TO ACT); /* Move current class qual values to active */
6089 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Return if bad status */
6090 1 ACT MRBPTR->MRB = CURR_MRBPTR->MRB; /* Move current MRB to active MRB */
6091 1 MRBPTR = ACT_MRBPTR; /* Make all MRB refs refer to "active" MRB */
6092 1 CALL = GET_CLASSES(AT LEAST ONE CLASS); /* Get info on requested classes */
6093 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Return if bad status */
6094 1 CALL = GET_QUALIFIERS(); /* Get the MONITOR command qualifiers */
6095 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Return if bad status */
6096 1
6097 1 IF AT LEAST_ONE_CLASS /* If at least one class requested, */
6098 1 THEN /*
6099 1     IF M->MRB$V MFSUM /* If this is a multi-file summary request, */
6100 1     THEN CALL = MFSUM_REQUEST(); /* Call special REQUEST routine for m.f. summary */
6101 1     ELSE DO; /* Regular MONITOR request */
6102 1         IF M->MRB$A_INPUT ^= NULL() /* If it's a playback, */
6103 1         THEN DO; /*
6104 1             IFBPTR = M->MRB$A_INPUT; /* Make MRB$A_INPUT point to a */
6105 1             M->MRB$A_INPUT = IFB$A_INPUT; /* ... file descr instead of an IFB */
6106 1             END;
6107 1
6108 1             IF M->MRB$A_SUMMARY ^= NULL() /* If summary requested, */
6109 1             THEN DO; /*
6110 1                 CALL = MOVE_CLASS_QUALS(ALL TO ACT); /* Force ALL stat value to active */
6111 1                 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Return if bad status */
6112 1                 END;
6113 1
6114 1                 CALL = EXECUTE_REQUEST(); /* Execute the MONITOR request */
6115 1                 END;
6116 1     ELSE /* Otherwise, no classes requested, */
6117 1         IF FIRST_MON_CMD = YES & QUAL_SPECIFIED = NO /* Check for bare MONITOR cmd at DCL */
6118 1         THEN PROMPT = YES; /* If so, user wants to go interactive */
6119 1         ELSE DO; /* MONITOR cmd issued without classes */
6120 1             CALL MON_ERR(MNRS_NOCLASSES); /* This is an error ... log it */
6121 1             RETURN(MNRS_NOCLASSES); /* ... and return */
6122 1             END;
6123 1 RETURN (CALL); /* Return with status code */
6124 1
6125 1 END MONITOR_CMD;
6126
```

```

6127 SET_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute a SET subcommand */
6128
6129 : 1 /*+++
6130 : 1 /*
6131 : 1 /* FUNCTIONAL DESCRIPTION:
6132 : 1 /*
6133 : 1 /* SET_CMD
6134 : 1 /*
6135 : 1 /* TBS
6136 : 1 /*
6137 : 1 /* INPUTS:
6138 : 1 /*
6139 : 1 /* TBS
6140 : 1 /*
6141 : 1 /* IMPLICIT INPUTS:
6142 : 1 /*
6143 : 1 /* TBS
6144 : 1 /*
6145 : 1 /* OUTPUTS:
6146 : 1 /*
6147 : 1 /* TBS
6148 : 1 /*
6149 : 1 /* IMPLICIT OUTPUTS:
6150 : 1 /*
6151 : 1 /* TBS
6152 : 1 /*
6153 : 1 /* ROUTINE VALUE:
6154 : 1 /*
6155 : 1 /* TBS
6156 : 1 /*
6157 : 1 /* SIDE EFFECTS:
6158 : 1 /*
6159 : 1 /* TBS
6160 : 1 /*
6161 : 1 /*
6162 : 1

```

```
6163 1
6164 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
6932 1
6933 1 %REPLACE NOT_SUCCESSFUL BY '0'B; /* Failing severity code */
6934 1
6935 1 /*
6936 1 /*
6937 1 /*
6938 1 /*
6939 1 /*
6940 1 /*
6941 1 /*/
6942 1
6943 1 Declare
6944 1 INP_PTR_VOL POINTER GLOBALDEF INIT(NULL()), /* Pointer to volatile /INPUT file-spec */
6945 1 DISP_PTR_VOL POINTER GLOBALDEF INIT(NULL()), /* Pointer to volatile /DISPLAY file-spec */
6946 1 REC_PTR_VOL POINTER GLOBALDEF INIT(NULL()), /* Pointer to volatile /RECORD file-spec */
6947 1 SUMM_PTR_VOL POINTER GLOBALDEF INIT(NULL()), /* Pointer to volatile /SUMMARY file-spec */
6948 1 COMM_PTR_VOL POINTER GLOBALDEF INIT(NULL()), /* Pointer to volatile /COMMENT string */
6949 1 IFB_TAB_VOL POINTER GLOBALDEF INIT(NULL()); /* Pointer to volatile IFB_TABLE */
6950 1
6951 1 Declare
6952 1 INP_PTR_SWAP BIT(1) ALIGNED GLOBALDEF, /* YES => swap INP_PTR_VOL and INP_PTR_PERM */
6953 1 DISP_PTR_SWAP BIT(1) ALIGNED GLOBALDEF, /* YES => swap DISP_PTR_VOL and DISP_PTR_PERM */
6954 1 REC_PTR_SWAP BIT(1) ALIGNED GLOBALDEF, /* YES => swap REC_PTR_VOL and REC_PTR_PERM */
6955 1 SUMM_PTR_SWAP BIT(1) ALIGNED GLOBALDEF, /* YES => swap SUMM_PTR_VOL and SUMM_PTR_PERM */
6956 1 COMM_PTR_SWAP BIT(1) ALIGNED GLOBALDEF, /* YES => swap COMM_PTR_VOL and COMM_PTR_PERM */
6957 1 IFB_TAB_SWAP BIT(1) ALIGNED GLOBALDEF; /* YES => swap IFB_TAB_VOL and IFB_TAB_PERM */
6958 1
6959 1 /*
6960 1 /*
6961 1 /*
6962 1 /*
6963 1 /*
6964 1 /*
6965 1 /*/
6966 1
6967 1 Declare
6968 1 MOVE_CLASS_QUALS ENTRY (FIXED BINARY(7)) /* MACRO-32 routine to move class ... */
6969 1 RETURNS(FIXED BINARY(31)), /* ... qualifier values for all classes */
6970 1 GET_CLASSES ENTRY (BIT(1) ALIGNED) RETURNS(FIXED BINARY(31)), /* Routine to get info on all classes for this req */
6971 1 GET_QUALIFIERS ENTRY RETURNS(FIXED BINARY(31)); /* Routine to store qualifier info for this request
6972 1
6973 1 Declare
6974 1 CUR_TO_ACT FIXED BINARY(31) GLOBALREF VALUE, /* Code value for MOVE_CLASS_QUALS rtn */
6975 1 ACT_TO_CUR FIXED BINARY(31) GLOBALREF VALUE, /* Code value for MOVE_CLASS_QUALS rtn */
6976 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
6977 1 CURR_MRBPTR POINTER GLOBALREF, /* Pointer to "current" MRB (Monitor Request Block) */
6978 1 TEMP_MRBPTR POINTER GLOBALREF; /* Pointer to "temp" MRB (Monitor Request Block) */
6979 1
6980 1 /*
6981 1 /*
6982 1 /*
6983 1 /*
6984 1 /*
6985 1 /*
```

MONMAIN
V04-000

K 15
16-SEP-1984 02:11:23
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (46) Page 64

```
6986 : 1  /*/
6987 1
6988 1 Declare
6989 1 INP_PTR_PERM POINTER STATIC INIT(NULL()), /* Pointer to permanent /INPUT file-spec */
6990 1 DISP_PTR_PERM POINTER STATIC INIT(NULL()), /* Pointer to permanent /DISPLAY file-spec */
6991 1 REC_PTR_PERM POINTER STATIC INIT(NULL()), /* Pointer to permanent /RECORD file-spec */
6992 1 SUMM_PTR_PERM POINTER STATIC INIT(NULL()), /* Pointer to permanent /SUMMARY file-spec */
6993 1 COMM_PTR_PERM POINTER STATIC INIT(NULL()), /* Pointer to permanent /COMMENT string */
6994 1 IFB_TAB_PERM POINTER STATIC INIT(NULL()); /* Pointer to permanent IFB_TABLE */
6995 1
6996 1 Declare
6997 1 CALL FIXED BINARY(31), /* Holds function value (return status) of called ro
6998 1 STATUS BIT(1) BASED(ADDR(CALL)), /* Low-order status bit for called routines */
6999 1 TEMP POINTER, /* Temporary "scratch" area */
7000 1 AT_LEAST_ONE_CLASS BIT(1) ALIGNED; /* YES => user requested at least one class */
7001 1
7002 1
```

```

7003 1 INP_PTR_SWAP = NO;
7004 1 DISP_PTR_SWAP = NO;
7005 1 REC_PTR_SWAP = NO;
7006 1 SUMM_PTR_SWAP = NO;
7007 1 COMM_PTR_SWAP = NO;
7008 1 IFB_TAB_SWAP = NO;
7009 1 TEMP_MRBPTR->MRB = CURR_MRBPTR->MRB;
7010 1 MRBPTR = TEMP_MRBPTR;
7011 1 CALL = GET_QUALIFIERS();
7012 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);
7013 1
7014 1 CALL = MOVE_CLASS_QUALS(CUR TO ACT);
7015 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);
7016 1
7017 1 CALL = GET_CLASSES(AT_LEAST_ONE_CLASS);
7018 1
7019 1 IF STATUS
7020 1 THEN DO;
7021 2 TEMP = TEMP_MRBPTR;
7022 2 TEMP_MRBPTR = CURR_MRBPTR;
7023 2 CURR_MRBPTR = TEMP;
7024 2
7025 2 IF INP_PTR_SWAP = YES
7026 2 THEN DO;
7027 3 TEMP = INP_PTR_VOL;
7028 3 INP_PTR_VOL = INP_PTR_PERM;
7029 3 INP_PTR_PERM = TEMP;
7030 3 END;
7031 2
7032 2 IF DISP_PTR_SWAP = YES
7033 2 THEN DO;
7034 3 TEMP = DISP_PTR_VOL;
7035 3 DISP_PTR_VOL = DISP_PTR_PERM;
7036 3 DISP_PTR_PERM = TEMP;
7037 3 END;
7038 2
7039 2 IF REC_PTR_SWAP = YES
7040 2 THEN DO;
7041 3 TEMP = REC_PTR_VOL;
7042 3 REC_PTR_VOL = REC_PTR_PERM;
7043 3 REC_PTR_PERM = TEMP;
7044 3 END;
7045 2
7046 2 IF SUMM_PTR_SWAP = YES
7047 2 THEN DO;
7048 3 TEMP = SUMM_PTR_VOL;
7049 3 SUMM_PTR_VOL = SUMM_PTR_PERM;
7050 3 SUMM_PTR_PERM = TEMP;
7051 3 END;
7052 2
7053 2 IF COMM_PTR_SWAP = YES
7054 2 THEN DO;
7055 3 TEMP = COMM_PTR_VOL;
7056 3 COMM_PTR_VOL = COMM_PTR_PERM;
7057 3 COMM_PTR_PERM = TEMP;
7058 3 END;

```

```

/* Ind no swap needed for 2 ptrs to /INPUT file-spec
/* Ind no swap needed for 2 ptrs to /DISPLAY file-sp
/* Ind no swap needed for 2 ptrs to /RECORD file-spe
/* Ind no swap needed for 2 ptrs to /SUMMARY file-sp
/* Ind no swap needed for 2 ptrs to /COMMENT string
/* Ind no swap needed for 2 ptrs to IFB_TABLE */
/* Move "current" MRB to "temp" MRB */
/* Make all MRB refs refer to "temp" MRB */
/* Get SET qualifiers */
/* Return if bad status */

/* Move current class qual values to active */
/* Return if bad status */

/* Get info on requested classes */

/* If successful, */

/* Swap "current" and ... */
/* ... "temp" MRB pointers */
/* .... */

/* If pointer swap required, */

/* Swap volatile and ... */
/* ... permanent /INPUT file-spec */
/* ... pointers */

/* If pointer swap required, */

/* Swap volatile and ... */
/* ... permanent /DISPLAY file-spec */
/* ... pointers */

/* If pointer swap required, */

/* Swap volatile and ... */
/* ... permanent /RECORD file-spec */
/* ... pointers */

/* If pointer swap required, */

/* Swap volatile and ... */
/* ... permanent /SUMMARY file-spec */
/* ... pointers */

/* If pointer swap required, */

/* Swap volatile and ... */
/* ... permanent /COMMENT string */
/* ... pointers */

```

MONMAIN
V04-000

M 15
16-SEP-1984 02:11:24
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONTOR.SRC]MONMAIN.PLI;1 (47)

Page 66

```
7059      2      IF IFB_TAB_SWAP = YES      /* If pointer swap required, */
7060      2      THEN DO;
7061      2          TEMP = IFB_TAB_VOL;      /* Swap volatile and ... */
7062      3          IFB_TAB_VOL = IFB_TAB_PERM;      /* ... permanent IFB_TABLE */
7063      3          IFB_TAB_PERM = TEMP;      /* ... pointers */
7064      3      END;
7065      3
7066      2
7067      2      CALL = MOVE_CLASS_QUALS(ACT TO CUR);      /* Move active class qual values back to current */
7068      2      IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);      /* Return if bad status */
7069      2
7070      2      END;
7071      1
7072      1      RETURN (CALL);      /* Return with status code */
7073      1
7074      1      END SET_CMD;
7075
```

```

7076 INIT_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute an INITIALIZE subcommand */
7077
7078 /*++
7079 /*
7080 /* FUNCTIONAL DESCRIPTION:
7081 /*
7082 /*     INIT_CMD
7083 /*
7084 /*     TBS
7085 /*
7086 /* INPUTS:
7087 /*
7088 /*     TBS
7089 /*
7090 /* IMPLICIT INPUTS:
7091 /*
7092 /*     TBS
7093 /*
7094 /* OUTPUTS:
7095 /*
7096 /*     TBS
7097 /*
7098 /* IMPLICIT OUTPUTS:
7099 /*
7100 /*     TBS
7101 /*
7102 /* ROUTINE VALUE:
7103 /*
7104 /*     TBS
7105 /*
7106 /* SIDE EFFECTS:
7107 /*
7108 /*     TBS
7109 /*
7110 /*/
7111

```

```

7112 : 1 /*
7113 : 1 /*
7114 : 1 /*
7115 : 1 /*
7116 : 1 /*
7117 : 1 /*
7118 : 1 /*
7119 : 1 /*
7120 : 1 Declare
7121 : 1 CALL          FIXED BINARY(31);          /* Holds function value (return status) of called ro
7122 : 1
7123 : 1
7124 : 1 /*
7125 : 1 /*
7126 : 1 /*
7127 : 1 /*
7128 : 1 /*
7129 : 1 /*
7130 : 1 /*
7131 : 1
7132 : 1 %INCLUDE      MONDEF;                      /* Monitor utility structure definitions */
7900 : 1
7901 : 1 Declare
7902 : 1 DEF_TO_CUR    FIXED BINARY(31) GLOBALREF VALUE;      /* Code value for MOVE_CLASS_QUALS rtn */
7903 : 1
7904 : 1 Declare
7905 : 1 MOVE_CLASS_QUALS  ENTRY (FIXED BINARY(7))          /* MACRO-32 routine to move class ... */
7906 : 1                   RETURNS(FIXED BINARY(31));        /* ... qualifier values for all classes */
7907 : 1
7908 : 1 Declare
7909 : 1 CURR_MRBPTR    POINTER GLOBALREF;                  /* Pointer to "current" MRB (Monitor Request Block)
7910 : 1 DEF_MRBPTR     POINTER GLOBALREF;                  /* Pointer to "default" MRB (Monitor Request Block)
7911 : 1

```

MONMAIN
V04-000

C 16
16-SEP-1984 02:11:25 VAX-11 PL/I X2.1-273 Page 69
5-SEP-1984 15:09:57 ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (50)

```
7912 1  CURR_MRBPTR->MRB = DEF_MRBPTR->MRB;  
7913 1  CALL = MOVE_CLASS_QUALS(DEF_TO_CUR);  
7914 1  
7915 1  RETURN (CALL);  
7916 1  
7917 1  END INIT_CMD;  
7918
```

```
/* Move default MRB to current MRB */  
/* ... and default class qual values to current */  
/* Return with status code from MOVE_CLASS_QUALS */
```

```

7919      HELP_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute a HELP subcommand */
7920      1
7921      1      /*+++
7922      1      /*
7923      1      /* FUNCTIONAL DESCRIPTION:
7924      1      /*
7925      1      /*      HELP_CMD
7926      1      /*
7927      1      /*      TBS
7928      1      /*
7929      1      /* INPUTS:
7930      1      /*
7931      1      /*      TBS
7932      1      /*
7933      1      /* IMPLICIT INPUTS:
7934      1      /*
7935      1      /*      TBS
7936      1      /*
7937      1      /* OUTPUTS:
7938      1      /*
7939      1      /*      TBS
7940      1      /*
7941      1      /* IMPLICIT OUTPUTS:
7942      1      /*
7943      1      /*      TBS
7944      1      /*
7945      1      /* ROUTINE VALUE:
7946      1      /*
7947      1      /*      TBS
7948      1      /*
7949      1      /* SIDE EFFECTS:
7950      1      /*
7951      1      /*      TBS
7952      1      /*
7953      1      /*/
7954      1

```

```
7955 1 %INCLUDE $HLPDEF; /* HELP flag definitions */
7992 1 %REPLACE NOT_SUCCESSFUL BY '0'B; /* Failing severity code */
7993 1 %REPLACE MON_HELP_LIB BY 'MNRHELP'; /* Log name for MONITOR subcommand help library */
7994 1 %REPLACE HELP_PARM BY 'HELP_KEYS'; /* Name of parameter for HELP command (rest of line)
7995 1
7996 1 /*
7997 1 /*
7998 1 /*
7999 1 /* LOCAL STORAGE
8000 1 /*
8001 1 /*
8002 1 /*/
8003 1 Declare
8004 1 CALL FIXED BINARY(31), /* Holds function value (return status) of called ro
8005 1 STATUS BIT(1) BASED(ADDR(CALL)); /* Low-order status bit for called routines */
8006 1
8007 1 /*
8008 1 /*
8009 1 /*
8010 1 /* EXTERNAL REFERENCES
8011 1 /*
8012 1 /*
8013 1 /*/
8014 1
8015 1 Declare
8016 1 CLISGET_VALUE ENTRY (CHAR(*), ANY) /* CLE routine to get qualifier values */
8017 1 RETURNS(BIT(1)),
8018 1 LBR$OUTPUT_HELP ENTRY (ENTRY VALUE, ANY, ANY, CHAR(*), FIXED BINARY(31), ENTRY VALUE)
8019 1 OPTIONS(VARIABLE) RETURNS(FIXED BINARY(31)), /* Rtn to output help info with prompting */
8020 1 LIB$PUT_OUTPUT ENTRY, /* Routine to write a line to terminal */
8021 1 LIB$GET_INPUT ENTRY, /* Routine to read terminal */
8022 1 MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
8023 1 /* Routine to log synchronous errors */
8024 1 MNR$HELPERR FIXED BINARY(31) GLOBALREF VALUE; /* Error message code */
8025 1
8026 1 Declare
8027 1 NORMAL FIXED BINARY(31) GLOBALREF, /* MONITOR normal return status */
8028 1 CURR_ERRCODE FIXED BINARY(31) GLOBALREF; /* MONITOR error status code currently expected */
8029 1
8030 1 Declare
8031 1 DYN_STRING GLOBALREF, /* Dynamic string descriptor */
8032 1 2 L FIXED BINARY(15), /* Length */
8033 1 2 TC CHAR(2), /* Type and Class */
8034 1 2 A POINTER; /* Address */
8035 1
8036 1
```

```

8037 1
8038 1      CURR_ERRCODE = MNRS_HELPERR;
8039 1      IF ^CLISGET_VALUE(HELP_PARM,DYN_STRING)
8040 : 1
8041 1          THEN DYN_STRING.L = 0;
8042 1
8043 1      CALL = LBR$OUTPUT_HELP(LIB$PUT_OUTPUT,,DYN_STRING,MON_HELP LIB,
8044 1          HLP$M_PROMPT+HLP$M_PROCESS+HLP$M_GROUP+HLP$M_SYSTEM+HLP$M_HELP,
8045 1          LIB$GET_INPUT);
8046 1
8047 1      IF STATUS = NOT_SUCCESSFUL
8048 1          THEN DO;
8049 2          CALL MON_ERR(MNRS_HELPERR,CALL);
8050 2          RETURN(MNRS_HELPERR);
8051 2          END;
8052 1
8053 1      RETURN (NORMAL);
8054 1
8055 1      END HELP_CMD;
8056 1

```

/* Set MONITOR code for signaled errors */
/* Get all command line after 'HELP' verb */
/* ... then ship it off to LBR\$OUTPUT_HELP */
/* If rest of line null, make string null */
/* Call standard HELP output routine */
/* If failed, */
/* Log the error ... */
/* ... and return with status */
/* Return with success code */

```

8057 EXIT_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute an EXIT subcommand */
8058
8059 /*++
8060 /*
8061 /* FUNCTIONAL DESCRIPTION:
8062 /*
8063 /*     EXIT_CMD
8064 /*
8065 /*     TBS
8066 /*
8067 /* INPUTS:
8068 /*
8069 /*     TBS
8070 /*
8071 /* IMPLICIT INPUTS:
8072 /*
8073 /*     TBS
8074 /*
8075 /* OUTPUTS:
8076 /*
8077 /*     TBS
8078 /*
8079 /* IMPLICIT OUTPUTS:
8080 /*
8081 /*     TBS
8082 /*
8083 /* ROUTINE VALUE:
8084 /*
8085 /*     TBS
8086 /*
8087 /* SIDE EFFECTS:
8088 /*
8089 /*     TBS
8090 /*
8091 /*/
8092

```

VAX-11 PL/I X2.1-273 Page 74
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (55)

**Declare
PROMPT
NORMAL**

```

BIT(1) ALIGNED GLOBALREF,
FIXED BINARY(31) GLOBALREF;

```

```
/* YES => prompt user for another subcommand */
/* MONITOR normal return status */
```

MONMAIN
V04-000

I 16
16-SEP-1984 02:11:27 VAX-11 PL/I X2.1-273 Page 75
5-SEP-1984 15:09:57 ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (56)

8115 1 PROMPT = NO;
8116 1
8117 1 RETURN (NORMAL);
8118 1
8119 1 END EXIT_CMD;
8120
8121

/* Indicate no more prompting for subcommands */

/* Return with status code */

```

8122 EXECUTE_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute a MONITOR command file */
8123
8124 /*++
8125 /*
8126 /* FUNCTIONAL DESCRIPTION:
8127 /*
8128 /* EXECUTE_CMD
8129 /*
8130 /* This is the routine invoked by CLISDISPATCH when the EXECUTE command is given to MONITOR
8131 /* at the interactive level. EXECUTE_CMD opens the execute command file, sets the bit flag EXECUTE,
8132 /* and returns to the main interactive loop in MONMAIN (which in turn calls NEXT_EXECUTE_COMMAND).
8133 /* If there is an error opening the file, the error is reported and control returns
8134 /* to the interactive level.
8135 /*
8136 /* INPUTS:
8137 /*
8138 /* Execute filename - this is the file that will be opened so NEXT_EXECUTE_COMMAND can read monitor
8139 /* commands. The filename is obtained by using CLISGET_VALUE with the CLD label 'EXEC_FILE'.
8140 /*
8141 /* IMPLICIT INPUTS:
8142 /*
8143 /* EXECUTE - a bit set to tell the main loop to call NEXT_EXECUTE_COMMAND instead of EXECUTE_COMMAND.
8144 /* NEXET_EXECUTE_COMMAND reads MONITOR commands from the opened file instead of the terminal.
8145 /*
8146 /* OUTPUTS:
8147 /*
8148 /* None.
8149 /*
8150 /* IMPLICIT OUTPUTS:
8151 /*
8152 /* COMMAND FILE - the file reference associated with the execute command file. This is also used by
8153 /* NEXT_EXECUTE_COMMAND to read the file.
8154 /*
8155 /* ROUTINE VALUE:
8156 /*
8157 /* MNRS_ERREXECOM - EXECUTE subcommand cannot be used in an execute command file.
8158 /* MNRS_ERREXEFIL - CLISGET_VALUE failed to return the execute file name.
8159 /* MNRS_ERREXEOPN - Open failure using the execute file name.
8160 /* SSS_NORMAL - Success.
8161 /*
8162 /* SIDE EFFECTS:
8163 /*
8164 /* None.
8165 /*
8166 /*/
8167

```

```

8168 | 1 /*
8169 | 1 /*
8170 | 1 /*
8171 | 1 /*
8172 | 1 /*
8173 | 1 /*
8174 | 1 /*
8175 | 1 /*
8176 | 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
8944 | 1
8945 | 1 %REPLACE FILE_SPEC_SIZE BY 128; /* Max file spec size */
8946 | 1 %REPLACE NOT_SUCCESSFUL BY '0'B; /* Failing severity code */
8947 | 1
8948 | 1
8949 | 1 /*
8950 | 1 /*
8951 | 1 /*
8952 | 1 /*
8953 | 1 /*
8954 | 1 /*
8955 | 1 /*
8956 | 1
8957 | 1 Declare
8958 | 1 COMMAND_FILE FILE RECORD GLOBALDEF; /* Execute command file */
8959 | 1
8960 | 1 /*
8961 | 1 /*
8962 | 1 /*
8963 | 1 /*
8964 | 1 /*
8965 | 1 /*
8966 | 1 /*
8967 | 1
8968 | 1 Declare
8969 | 1 CLISGET_VALUE ENTRY(ANY, ANY) /* CLE routine to get qualifier values */
8970 | 1 RETURNS(FIXED BINARY(31)),
8971 | 1 MON_ERR ENTRY (ANY VALUE, ANY, ANY) /* Routine to log synchronous errors */
8972 | 1 OPTIONS(VARIABLE);
8973 | 1 Declare
8974 | 1 CURR_ERRCODE FIXED BINARY(31) GLOBALREF, /* MONITOR error status code currently expected */
8975 | 1 EXECUTE BIT(1) ALIGNED GLOBALREF, /* YES => read another command from the execute file
8976 | 1 NORMAL FIXED BINARY(31) GLOBALREF; /* MONITOR normal return status */
8977 | 1
8978 | 1 Declare
8979 | 1 MNRS_ERREXECOM FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
8980 | 1 MNRS_ERREXEFIL FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
8981 | 1 MNRS_ERREXEOPN FIXED BINARY(31) GLOBALREF VALUE; /* Error message code */
8982 | 1
8983 | 1
8984 | 1 /*
8985 | 1 /*
8986 | 1 /*
8987 | 1 /*
8988 | 1 /*
8989 | 1 /*
8990 | 1 /*

```

```
8991 1
8992 1
8993 1   Declare
8994 1       CALL          FIXED BINARY(31),
8995 1       STATUS        BIT(1) BASED(ADDR(CALL));          /* Holds function value (return status) of called ro
8996 1
8997 1   Declare
8998 1       1 EXEC_FILE_PARM,
8999 1           2 L          FIXED BINARY (31),
9000 1           2 A          POINTER,
9001 1
9002 1       1 EXEC_FILE_VAL,
9003 1           2 L          FIXED BINARY (31),
9004 1           2 A          POINTER,
9005 1
9006 1       EXEC_FILE_NAME  CHAR(9)  STATIC INIT('EXEC FILE'),
9007 1       EXEC_FILE_STR   CHAR(FILE_SPEC_SIZE) STATIC INIT('MONITOR.MON');
9008 1
9009 1   ON UNDEFINEDFILE (COMMAND_FILE) GOTO OPEN_ERROR;          /* Set up the UNDEFINEDFILE condition */
9010 1
9011 1
9012 1   IF EXECUTE = YES
9013 1       THEN DO;
9014 2       CALL MON ERR(MNRS_ERREXECOM);
9015 2       RETURN(MNRS_ERREXECOM);
9016 2       END;
9017 1
9018 1   EXECUTE = YES;
9019 1   EXEC_FILE_PARM.L = LENGTH(EXEC_FILE_NAME);
9020 1   EXEC_FILE_PARM.A = ADDR(EXEC_FILE_NAME);
9021 1   EXEC_FILE_VAL.L = LENGTH(EXEC_FILE_STR);
9022 1   EXEC_FILE_VAL.A = ADDR(EXEC_FILE_STR);
9023 1   CURR_ERRCODE = MNRS_ERREXEFIL;
9024 1   CALL = CLISGET VALUE(EXEC_FILE_PARM,EXEC_FILE_VAL);
9025 1   IF STATUS = NOT_SUCCESSFUL
9026 1       THEN DO;
9027 2       EXECUTE = NO;
9028 2       CALL MON ERR(MNRS_ERREXEFIL,CALL);
9029 2       RETURN(MNRS_ERREXEFIL);
9030 2       END;
9031 1
9032 1   OPEN FILE(COMMAND_FILE) INPUT SEQUENTIAL TITLE(EXEC_FILE_STR)
9033 1       ENVIRONMENT(DEFAULT_FILE_NAME('MON'));
9034 1
9035 1   CURR_ERRCODE = 0;
9036 1   RETURN (NORMAL);
9037 1
9038 1   OPEN_ERROR:
9039 1       EXECUTE = NO;
9040 1       CURR_ERRCODE = 0;
9041 1       CALL MON ERR(MNRS_ERREXEOPN);
9042 1       RETURN(MNRS_ERREXEOPN);
9043 1
9044 1   END EXECUTE_CMD;
```

COMMAND LINE

MONMAIN
V04-000

M 16
16-SEP-1984 02:11:29
5-SEP-1984 15:09:57

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]MONMAIN.PLI;1 (58)

Page 79

PLI/LIS=LIS\$:MONMAIN/OBJ=OBJ\$:MONMAIN MSRC\$:MONMAIN+LIB\$:MONLIB/LIB

0241

AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

0242 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

MONMSG
LIS

REQUEST
LIS

SHODEF
LIS

MONSUB
LIS

PREPOST
LIS

SUMMBUFF
LIS